NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A REAL-TIME SYSTEM FOR ABUSIVE NETWORK TRAFFIC DETECTION

by

Georgios Kakavelakis

March 2011

Thesis Advisor:	Robert Beverly
Second Reader:	Joel D. Young

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2011	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Real-Time System for Abusive Network Traffic Detection		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Georgios Kakavelakis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number _____N/A_____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

Abusive network traffic—to include unsolicited e-mail, malware propagation, and denial-of-service attacks—remains a constant problem in the Internet. Despite extensive research in, and subsequent deployment of, abusive-traffic-detection infrastructure, none of the available techniques addresses the problem effectively or completely. The fundamental failing of existing methods is that spammers and attack perpetrators rapidly adapt to and circumvent new mitigation techniques. Analyzing network traffic by exploiting transport-layer characteristics can help remedy this and provide effective detection of abusive traffic.

Within this framework, we develop a real-time, online system that integrates transport layer characteristics into the existing SpamAssasin tool for detecting unsolicited commercial e-mail (spam). Specifically, we implement the previously proposed, but undeveloped, SpamFlow technique. We determine appropriate algorithms based on classification performance, training required, adaptability, and computational load. We evaluate system performance in a virtual test bed and live environment and present analytical results. Finally, we evaluate our system in the context of SpamAssassin's auto-learning mode, providing an effective method to train the system without explicit user interaction or feedback.

14. SUBJECT TERMS Network Security, Autonomous Systems, Machine Learning			**15. NUMBER OF PAGES** 89
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A REAL-TIME SYSTEM FOR ABUSIVE NETWORK TRAFFIC DETECTION

Georgios Kakavelakis
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2011

Author: Georgios Kakavelakis

Approved by: Robert Beverly
Thesis Advisor

Joel D. Young
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Abusive network traffic—to include unsolicited e-mail, malware propagation, and denial-of-service attacks—remains a constant problem in the Internet. Despite extensive research in, and subsequent deployment of, abusive-traffic-detection infrastructure, none of the available techniques addresses the problem effectively or completely. The fundamental failing of existing methods is that spammers and attack perpetrators rapidly adapt to and circumvent new mitigation techniques. Analyzing network traffic by exploiting transport-layer characteristics can help remedy this and provide effective detection of abusive traffic.

Within this framework, we develop a real-time, online system that integrates transport layer characteristics into the existing SpamAssasin tool for detecting unsolicited commercial e-mail (spam). Specifically, we implement the previously proposed, but undeveloped, SpamFlow technique. We determine appropriate algorithms based on classification performance, training required, adaptability, and computational load. We evaluate system performance in a virtual test bed and live environment and present analytical results. Finally, we evaluate our system in the context of SpamAssassin's auto-learning mode, providing an effective method to train the system without explicit user interaction or feedback.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AS	Autonomous System
BGP	Border Gateway Protocol
CSS	Cascading Style Sheets
DCC	Distributed Checksum Clearinghouses
DKIM	DomainKeys Identified Mail Signatures
DNS	Domain Name System
DoS	Denial of Service
FIN	Finish (flag)
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Provider
MAP	Maximum A-posteriori Probability
MTA	Message Transfer Agent
OS	Operating System
RAM	Random Access Memory
RPC	Remote Procedure Call
RST	Reset (flag)
RTT	Round-trip Time
SMTP	Simple Mail Transfer Protocol
SYN	Synchronize (flag)
TCP	Transmission Control Protocol
TOS	Type of Service
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my thesis advisor, Professor Robert Beverly, for his patient guidance and continual encouragement and my second reader, Professor Joel D. Young, for his valuable insights and comments. Without their knowledge and assistance, this thesis would not have been successful.

Above all, I would like to express my love and gratitude to my wife, Dimitra, and my daughter, Afroditi, for their understanding and endless love through the duration of my studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A major use of the Internet is trade and e-commerce, and increased reliance on the Internet for these functions demands increased reliability and security. The increase in Internet use has led to the evolution of technologies that permit high traffic and improved network performance with regard to bandwidth and traffic capacity. Abuse of the Internet infrastructure has unfortunately also increased, in the form of denial-of-service (DoS) attacks, worms, spam abusive traffic, DoS spam attacks, and so on. Internet abuse is increasing sociologically as well as technologically, with organized criminals and other malicious individuals exploiting the potential of network abuse.

A. SCOPE

The scope of this thesis is to: i) develop a real-time, online system, based on the previous work of [3], that detects abusive network traffic associated with unsolicited commercial e-mail, aka spam; ii) determines the most appropriate algorithms for such a detector; iii) evaluates its performance; and iv) presents analytical results from running the system.

Electronic mail (e-mail) is one of the most popular applications of the Internet, enabling users to easily communicate by exchanging electronic messages at no upfront cost, quickly, reliably and easily. E-mail distribution relies on an infrastructure consisting of three components: user agents, mail servers, and Simple Mail Transfer Protocol (SMTP) [4]. User agents allow users to read, reply to, forward, save, and compose messages, whereas mail servers or message-transfer agents (MTAs) [5] are the core of the e-mail infrastructure, responsible for the proper store-and-forward dissemination of electronic mail. SMTP is the application protocol normally used for e-mail exchange and leverages the reliable transfer properties of TCP [6] to deliver mail from the sender's MTA to the recipient's MTA [7]. The e-mail architecture of the Internet is over three decades old and was designed at a time when the implicit assumption was that a user wanted to receive all messages addressed to him.

Unfortunately, this popular communication media has been exploited. E-mail abuse includes using high volumes of e-mail to distribute various types of content (as shown in Figure 1) ranging from product advertisements to malware and pornography, delivered to unsuspecting clients without their consent. These kinds of messages are known as unsolicited commercial e-mail or "spam." Abusive e-mail started to become a problem when the Internet was opened to the public and has increased from approximately 10% of overall e-mail volume in 1998 to a fairly consistent rate of about 88% to 92% today, posing a great burden not only to users but to service providers, companies, and the network itself [8].

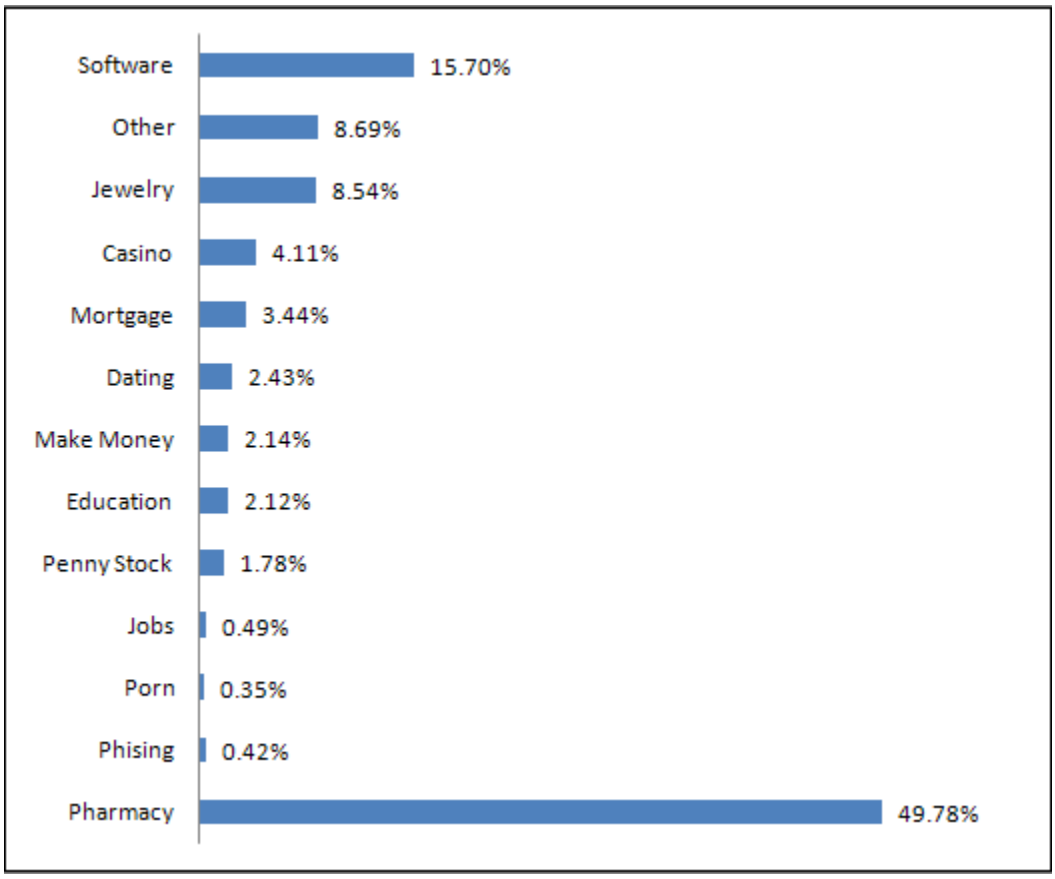

Figure 1. Spam E-mail Detected [From [1]]

The nuisance factor of spam is manifold. It is irritating to sift cautiously through quantities of junk e-mail to find legitimate messages, and it is a waste of time and productivity. Besides that, spam may violate user privacy, for example, by phishing, in which the spammer deceives the recipient by pretending to be a trusted party and asks for sensitive information (passwords, credit-card numbers, etc.) [9]. Spam e-mail is a problem for mail providers because it reduces storage space and consumes computational resources [10]. Network performance is degraded, since bandwidth is wasted in delivering spam e-mail and congestion increases on the links.

Many methods have been proposed to address the increasing problem of spam. One of the earliest, still used today, rejects messages that originated from senders found in blacklists—that is, databases, such as Spamhaus [11] and MAPS [12], that contain untrusted IP [13] addresses. Content filtering, another popular technique, relies on the assumption that spam messages contain words or phrases that differentiate them from legitimate e-mail messages. Systems that use this technique check the body and headers of a message for indicative words or phrases, using either a rule- or learning-based approach. Rule-based systems are less effective because the user has to be involved in the construction and update of the rules, which is time consuming and error prone. By contrast, learning-based systems use machine-learning algorithms to automatically categorize a message as spam or legitimate. These systems need to be trained on a set of messages in order to extract the features, words, or phrases that will become the basis for classifying messages. Spammers, meanwhile, adapt accordingly and find countermeasures, such as fake IP addresses or compromised hosts, also known as botnets, to evade blacklisting. To counter content filtering, they use sophisticated HTML- and CSS-based obfuscation techniques or place the entire message content in randomized images [14].

Traffic-characterization studies [3, 15, 16] try to address these issues by examining network characteristics associated with spam behavior at the IP and TCP level. Studies have shown that spam messages frequently originate from specific IP-address space regions and autonomous-system (AS) numbers. To be more effective and hide their trails, spammers take advantage of compromised hosts to send unsolicited

commercial e-mail, which manifests itself in specific TCP features such as packet drops, retransmissions, and variable roundtrip times (RTT) [6]. These techniques are promising since it is more difficult for spammers to thwart such characteristics by manipulating the IP or TCP layer.

The reputation of senders, messages, or flows, and collaboration among systems and providers can leverage the above techniques and provide a more holistic view of the methods and behavior that spammers use to obscure themselves.

B. GOALS

The goals of this thesis are summarized as follows:

- Develop a test bed consisting of three infrastructure components: i) the user agent implementing the client side of the SMTP protocol and generating e-mail traffic; ii) the network emulator, which mimics the network path and condition characteristics to create traffic analogous with that of a live network; and iii) the MTA, which implements the server side of SMTP protocol.
- Develop the user agent, which will take as input messages from a corpus and replay them in such a way that we can differentiate spam from legitimate traffic and establish a ground truth when we receive messages on the MTA side.
- Modify the network emulator so that it can produce random delay with mean μ and standard deviation σ. Create two different traffic schemes: one that simulates legitimate traffic and one that simulates spam-traffic characteristics, such as loss of packets, retransmissions, and variable RTT.
- Modify our MTA to include the port number along with the IP address of the sender in the message headers; the (IP:Port) tuple will be our message identifier.
- Integrate SpamAssassin with our MTA; SpamAssassin will be the host of our real-time system.

- Integrate our system with SpamAssassin by developing a plug-in (*controller*) that will control both the *flow-analysis engine* and the *classification engine*. Further, it will aggregate data from our classification engines and build the confusion matrices that will be used later in the evaluation process.
- Develop our classification engine in a simple and extensible way by utilizing existing technologies such as the classifiers that are provided by Orange, a statistical- and machine-learning software package, and the XML-RPC protocol for the establishment of two-way communication between *controller* and the *classification engine.*
- Evaluate the performance of our system within the test bed with respect to accuracy, precision, and recall.
- Deploy our system in a live environment and evaluate its performance.
- Evaluate how our system performs in an auto-learning fashion. We describe how we define auto-learning and discuss the results in Chapter IV, Section C.
- Discuss future work, such as other fields of abusive traffic where our system may be used and enhancements of the existing system.

C. MAJOR RESULTS

The major results of this thesis are summarized in the following points:

- Our system achieved greater than 90% accuracy, precision, and recall in both the virtual test bed and live environment, independent of the classification method—which indicates that it can adopt and capture any changes in TCP characteristics.
- We achieved a 99% precision rate in live testing with as few as 128 training examples, which suggests that spam flows are characterized by high entropy, and we achieve small initialization times.

- In auto-learning mode with SVM, we achieved above 97% rates in accuracy and precision and above 99% in recall, with as few as 256 training examples.

D. STRUCTURE

The structure of our thesis is as follows: in Chapter II, we present previous work in the field of spam filtering and detection and discuss the machine-learning algorithms that we use for our classification engines. We elaborate on the test-bed architecture and system design in Chapter III, and in Chapter IV discuss our experiments within the test bed and the live environment, along with the evaluation results. Finally, in Chapter V we summarize our work and its results and discuss future work having to do with evaluation, the detection of other types of abusive traffic, unsupervised learning, and system enhancements that will increase usability.

II. RELATED WORK

Many methods have been proposed to address the problem of spam. Among the most popular and widely deployed are content filtering, collaborative filtering, and reputation systems. We review these first, then discuss emerging work in traffic characterization, which is most relevant to this thesis.

A. CONTENT FILTERING

Content filters are founded on the premise that spam and legitimate e-mail contain features, in this case, words, that are statistically distinguishable. In general, a filter [17] is a function that takes as input the message to be classified and a model, and outputs a classification label.

$$f(m, M) = \begin{Bmatrix} c_{spam}, \text{ if the decision is spam} \\ c_{leg}, \text{ otherwise} \end{Bmatrix},$$

In the context of messaging and binary labels (spam or legitimate), m is the message to be classified, M is the model, and c_{spam} and c_{leg} are the classification labels assigned to the message.

The model can be either rule or learning based. Rule-based models consist of logical rules that have to be continuously updated and refined by the user in order to be competent with the dynamic nature of spam e-mail. Updating rules is problematic because it is a time-consuming and often error-prone process [18]. In learning-based approaches, the model is the outcome of applying a training algorithm on the features of a selected set of labelled training messages. The objective is to create a model that generalizes to predicting the classification of new, unseen messages. Each message is mapped to a feature vector x composed of message characteristics, either textual or nontextual, from a dictionary formed by analyzing the messages. For textual features [8], we consider individual words, particular phrases, or overemphasized punctuation, such as "!!!." Nontextual features [8] can be the domain type of the message sender (e.g., .edu or .com), whether the message was sent via a mailing list, or whether it has an attached

document (most junk e-mail does not, but some malware propagates via e-mail). Feature vectors can be constructed by various methods. We mention some of them [19]:

- term frequency, where each feature is represented by the number of times that it appears in a given message (often normalized by frequency across messages)
- binary representation, which indicates whether a particular feature occurs in the message
- use of a stop list as a supplementary method to the above. The stop list contains words like “a,” “and,” “the,” etc., that are not used in the forming of the feature vector.
- use of stemming as a supplementary method. Stemming reduces words to their root, for instance “builder,” “build,” and “building” share a common root. This technique also makes for a more compact model representation, while increasing accuracy.

Learning-based filters have been the focus of considerable interest and one can select from a wide variety of machine-learning algorithms. We further elaborate on the three that we used to evaluate our system.

1. Naïve Bayesian Classifier

Naïve Bayes classifiers [20, 21] were used in [18, 22] as an automated method for filtering spam, in order to overcome the problems of manually constructing logical rules, which require users, on one hand, to be capable of constructing robust rules and, on the other, to constantly tune and refine the rules to adapt to the continuously changing nature of spam e-mail. Their experiments revealed impressive results on both precision and recall.

The naïve Bayes classifier is based on the Bayes theorem and the assumption that each feature is conditionally independent of every other feature, given the class variable *C*. The Bayes theorem is defined as [22]:

$$P\left(C = c_k \mid \vec{F} = \vec{f}\right) = \frac{P\left(\vec{F} = \vec{f} \mid C = c_k\right) P(C = c_k)}{P(\vec{F} = \vec{f})},$$

where C is the class variable and F is the feature vector. Applying the independence assumption:

$$P\left(\vec{F} = \vec{f} \mid C = c_k\right) = \prod_i P\left(\vec{F}_i = \vec{f}_i \mid C = c_k\right),$$

and by using the maximum a-posteriori probability (MAP), the basic decision rule can be defined as follows [17]:

$$\begin{aligned} c &= classify\left(f_1, f_2, ..., f_n\right) \\ &= \underset{k=\{spam,leg\}}{\operatorname{argmax}} \left(P\left(C = c_k \mid \vec{F} = \vec{f}\right)\right) \\ &= \underset{k=\{spam,leg\}}{\operatorname{argmax}} \left(\frac{P\left(\vec{F} = \vec{f} \mid C = c_k\right) P(C = c_k)}{P(\vec{F} = \vec{f})}\right) \\ &= \underset{k=\{spam,leg\}}{\operatorname{argmax}} \left(P(C = c_k) \prod_i P\left(\vec{F}_i = \vec{f}_i \mid C = c_k\right)\right) \end{aligned}$$

The prior probability $P\left(C = c\right)$ is given by the ratio of the number of examples that belong in class c to the total number of examples. The product of the conditional probabilities depends on the feature types, whether they are discrete or continuous. If the features are discrete, the conditional probability is the ratio of the number of vectors F_i that have value f_i and belong to class c_k to the total number of vectors that belong to class c_k. In the case of continuous values, we assume that they follow a normal distribution and we have:

$$P\left(\vec{F}_i = \vec{f}_i \mid C = c_k\right) = g\left(x_i; \mu_{i,c_k}, \sigma_{i,c_k}\right),$$

where

$$g\left(x; \mu_X, \sigma_X\right) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

is the normal (Gaussian) distribution.

2. Support Vector Machines

Support-vector machines were introduced by Vapnic [23]. The main objective of SVM is to discover the optimal hyperplane[1] in n-dimensional feature space, such that the feature vectors of each class exist on the same side of the hyperplane. For example, if we take a random-feature vector that is closer to the feature vectors of class c_1 than to c_2, it will reside in the hyperplane side that represents class c_1. Therefore, after the discovery of the optimal hyperplane, we will be able to correctly classify a given example.

A hyperplane ***P*** is considered optimal if it maximizes the minimum margin, i.e., if the distances of the closest feature vectors of each class from ***P*** are equal. Formally, we can represent ***P*** with the equation $\vec{w} \bullet \vec{x} + b = 0$, where $\vec{w}$ is the normal vector[2] of ***P***, b is a term that allows the algorithm to choose among all the hyperplanes that are perpendicular to the normal vector ***P,*** and $\bullet$ is the inner product. The space that separates the feature vectors of each class is defined as the margin between two hyperplanes with the following equations:

$$\vec{w} \bullet \vec{x}_i + b = 1, \text{ if } c_i = 1$$
$$\vec{w} \bullet \vec{x}_i + b = -1, \text{ if } c_2 = -1$$

Therefore, every training example belongs to class c_i, if $\vec{w} \bullet \vec{x}_i + b \geq 1$ and to class c_2, if $\vec{w} \bullet \vec{x}_i + b \leq 1$. Our goal is to maximize the margin. In that way, the classes will have a confident degree of separation, thus allowing us to make more effective classifications. So after having found the support vectors, the decision rule to classify an unknown example is the following:

$$f\left(\vec{x}\right) = sign\left(\sum_{i=1}^{n} a_i y_i x_i + b\right),$$

where a_i and b are used to maximize the margin of the separating hyperplane and $y_i = \{1, -1\}$ are the classes.

[1] A hyperplane in R^1 is a point, in R^2 is a line and in R^3 is a proper plane.

[2] Normal vector $\vec{w}$ is perpendicular to the hyperplane.

The vectors, however, are not always linearly separable, so we have to apply a transformation function (10) $\Phi : R^n \to F$ from the n-dimensional feature space to another feature space. In that case, the decision rule becomes (1):

$$f\left(\vec{x}\right) = sign\left(\sum_{i=1}^{n} a_i y_i K\left(\vec{x}_i, x\right) + b\right),$$

where $K\left(\vec{x}_i, \vec{x}_j\right) = \Phi\left(\vec{x}_i\right) \bullet \Phi\left(\vec{x}_j\right)$ is the kernel function.

3. C4.5

C4.5 [24] is an extension of the ID3 decision-tree algorithm, designed to address ID3 issues, such as the handling of continuous attributes, avoiding overfitting data, reducing error pruning, handling missing values, etc.

The algorithm evaluates an unknown feature vector based on the following strategy: initially, it selects the best feature as the root of the decision tree. For every different value of the feature, it creates a descendant node, which consists of all the vectors that contain the specific feature value. This whole process is repeated recursively for each feature node in the decision tree. The process ends when one of the following conditions is met:

1. all vectors of the current node belong to the same class or
2. all features are used

How well the decision tree will perform depends on the selection process of the best feature. This will allow us to have a better clustering for each class of examples. A suitable measure for the evaluation of the features, and therefore for the selection of the best feature, is the information gain (IG) of an attribute A [25]. If we define S as the set of training examples, then the mathematical representation of IG is given by the following formula:

$$IG(A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v), \text{ where}$$

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i,$$

Values(A) is the set of attribute A values, Sv is a subset of S that contains the examples

with attribute A having value v, and pi is the ratio of the number of examples that belong to class i to the total number of examples. Entropy represents the amount of information that is provided by the attribute and, in information theory, is measured in bits [25]. Our goal is to maximize the IG of the selected attribute by minimizing the entropy of Sv, or, in other words, by reducing the number of bits.

Information gain, however, has the disadvantage that it selects attributes with a large set of values. To overcome this shortcoming, Quinlan [26] suggests utilizing the information-gain ratio, which is formalized as follows:

$$GR(A) = \frac{IG(A)}{IV(A)}, \text{ where}$$

$$IV(A) = -\sum_{i=1}^{|A|} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|},$$

and Si are the subsets of S that contain attribute A with value i. So, again, our goal is to find the attribute that maximizes the above ratio.

Drucker et al. [19] evaluated both the SVM and C4.5 algorithms on the spam problem, reporting acceptable results in terms of accuracy. Furthermore, Blanzieri and Bryl [27] improved the accuracy of the SVM filtering technique by leveraging the phenomenon of locality in spam [17].

Spammers, however, can easily evade content filters through different techniques, such as misspelling words, inserting HTML tags inside words to avoid detection of typical spam keywords, or lately, encapsulating the spam message inside an attached image, better known as image-based spam [28].

Furthermore, the user is another factor that determines the performance of content filters. Users can give feedback to the systems by reporting false positives and false negatives in order to retrain the classifiers. The major concern with users is that their classification is subjective and subject to error. Every user has a different notion of which e-mail is spam. Most of them classify an e-mail as spam not objectively, based on the definition of spam as unsolicited commercial e-mail, but rather, subjectively, based on the fact that it has no interest to them [29].

B. COLLABORATIVE FILTERING

Collaborative filtering addresses the problem of the user's subjective assessment. This technique is based on the idea that if many users collaborate and share their subjective assessments of an e-mail, they can be leveraged to create a more objective classification on that specific e-mail. The larger the number of users that collaborate, the better the results will be. For example, if a user decides to report a message as non-spam when the application knows that 10,000 users have reported it as spam, the application will ignore his suggestion. But if the numbers of votes is small, the objectivity of the suggestion is in doubt.

In a collaborative system [29], whenever a user receives an e-mail, a filtering application suggests a classification for the e-mail: either spam or not. Then the user can decide whether to accept this suggestion or deny it. If the user classifies the message as spam, a signature is computed for that e-mail and is reported to a collective knowledge base. If the signature matches a known signature of the database, it is then regarded as spam.

The robustness of the filter depends greatly on the signature algorithm. Spammers, in order to evade collaborative-filtering techniques, change at random small portions of the message, with the intention of making each spam message unique. If the signature algorithm fails to ignore such small randomizations, it will produce different signatures for the same message. For greater robustness, algorithms have been developed to be more content aware, so that unimportant changes do not alter the signature. For example, Razor [30] uses short-lived signatures where the signature is based on text that is selected from the spam message, based on a random number that changes regularly.

C. REPUTATION SYSTEMS

All transactions on the Internet today are covered by the umbrella of relative anonymity. This allows users to act maliciously without any consequences. Reputation systems try to solve this problem by developing trustworthy relationships between producers and consumers. Their goal is to assign a reputation score to an e-mail entity.

For this to be accomplished, these systems collect feedback from users, create a behavioral profile, and assign a score based on previous behavior [31].

Reputation systems can be categorized [31] according to two characteristics: the method of identifying the sending entity and the type of feedback that is further processed.

1. Identification Method

Entity identification is accomplished by using either the content or the address of the message. Systems that are founded on content-based identifiers use a form of fingerprint to establish a good correlation between message and entity. We can define a fingerprint as a many-to-one mapping. A good fingerprinting algorithm must not be susceptible to message mutation. Thus, it must map many similar messages to the same fingerprint while not mapping any additional messages to the fingerprint [32]. Razor [30] and DCC [33] are two such systems that use fingerprinting to identify a message sender.

Address-based identifiers can be an IP address, sender domain, or the entire address of the message (IP and domain). Systems that use the IP address as an identifier have as a back end real-time databases, which query in order to find out whether the IP address is blacklisted. These systems can be considered binary reputation systems [38] since they do not give a score but a yes/no answer. The disadvantages of this method [31] are that a legitimate host can be compromised and used to send spam messages, its IP address can be hijacked, and legitimate users can share IP addresses with others that send spam. Furthermore, Ramachandran et al. [15] showed that as much as 35% of spam messages were sent from IP addresses that were not included in blacklists.

As mentioned, another type of address-based identifier that reputation systems can use is the sender domain. Sender-authentication schemes, e.g., SenderID [34] and DKIM [35], prevent domain spoofing. SenderID is a path-based technology in which domain owners publish DNS TXT records that indicate which IP addresses are allowed to send e-mail on behalf of a given domain. DKIM is a crypto-based technology. The sender signs the message with a private key associated with the domain and the recipient uses the public key advertised in the DNS to verify the sender domain [31].

2. Feedback Method

Reputation systems use two types of feedback: reactive and predictive.

We define reactive feedback [31] as the feedback provided by humans or automated means such as spamtraps, honeypots, or other filtering systems. Examples of such systems are real-time blacklists and collaborative-filtering systems like those mentioned above. We must, however, ensure that the feedback provided originates from legitimate sources. In other words, we must establish a reporter's trustworthiness, otherwise the data are susceptible to malicious or accidental pollution.

Predictive feedback has to do with building behavioral-feature vectors based on a vast amount of observed activity for given identifiers [31]. The behavioral characteristics can be extracted from statistical properties such as volume, frequency, and distribution of identifiers or relations among identifiers. An example of a system that uses statistical properties is DCC, which uses the message fingerprint as an identifier and measures the volume of reports for each fingerprint. If the volume exceeds a certain threshold, the message is considered spam. Further, Leiba et al. [36] assign a reputation score to a message based on statistics for each IP address of the SMTP path. The statistics are the number of spam or legitimate e-mails for which each IP address on the SMTP path appears. Goldbeck and Handler [37] use the social network of users and user-assigned reputation scores for people they know to build a large reputation network, from which they can infer recursively a reputation score for the sender of a message.

Reputation systems, however, face some difficulties. First of all, there is no standard to define what constitutes a reputation score, so each vendor uses different criteria; and second, there is no centralized clearinghouse of reputations, which makes it difficult for vendors to exchange reputation scores [38].

D. TRAFFIC CHARACTERIZATION

Traffic-characterization methods are a recent novel approach to differentiating sources of abusive traffic. Several prior works are directly relevant to our research. These methods try to identify spam by leveraging the network or transport-layer properties.

Whereas spammers have the ability to alter the content of a message or spoof an IP address or sender domain, they have much less power to forge network (e.g., IP) or transport-level (e.g., TCP) properties.

1. Network-Level Characteristics

Ramachandran et al. [15] examine the spamming behavior at the network layer (IP layer) by correlating data collected from three sources: a sinkhole, a large e-mail provider, and the "command and control" of a Bobax botnet. More specifically, they focused on the following network-level properties:

- IP address space from which spam originates
- autonomous systems that sent spam messages to their sinkhole
- BGP route announcements

With respect to IP address space, their findings showed that spam and legitimate e-mail originate from the same portion of the IP address space, suggesting that it is not a good discriminating property. Autonomous-system (AS) utilization, on the other hand, showed that spammers use different ASs to disseminate their load as compared to the ASs that legitimate e-mail is sent from—which suggests that it could be a promising feature for filtering systems of spam messages.

Hao et al. [16], however, showed that AS alone as a feature may cause a large rate of false positives. Their work focused on extracting lightweight features from network-level properties such as geodesic distance between sender and receiver, sender IP-neighborhood density, probability ratio of spam to ham at the time of day the packet arrives, the AS number of the sender, and the status of open ports on the sender machine. Their feature-selection process showed that AS is the most influential feature, but when used for classification yields a false-positive rate of 0.76% under a 70% detection rate, which suggests that it should be used in combination with other features. Further studies [39, 40] have shown that a spammer can evade this technique by advertising routes from a forged AS number [16].

2. Transport-Level Characteristics

In a spirit similar to Ramachandran et al., Beverly and Sollins [3] explored transport-layer characteristics in order to determine whether spam e-mail presents different behavior from legitimate e-mail. Their idea is based on the premise that spammers have to send large volumes of e-mail to be effective, which suggests that the network links involved would experience contention and congestion. Therefore, transport-layer properties such as number of lost segments and the roundtrip time (RTT) would have different metrics in such a contentious environment, allowing discrimination between spam and legitimate behavior.

The features that they used to evaluate the transport-layer properties are the following:

- number of packets
- retransmissions
- packets with RST bit set
- packets with FIN bit set
- number of times zero window was advertised
- number of times minimum window was advertised
- maximum idle time between packets
- initial roundtrip time estimate
- variance of inter-packet delay

Among those features, the feature-selection process showed that RTT and minimum-congestion window are the most discriminatory. Their analysis revealed that 50% of spam messages have an RTT greater than 200ms, which correlates with the findings of Hao et al. [16] that showed that spam messages travel longer distances than legitimate ones. As for the performance of the classifier, the evaluation showed that it exhibits more that 90% accuracy and precision.

Moreover, Ouyang et al. [41] conducted a large-scale empirical analysis of transport-layer characteristics on 600K+ messages, based on the work of Beverly and Sollins. They expanded the feature set to include other features such as the operating

system of the remote host, the advertised window size in the SYN packet from the remote host, and variance of RTT. Among the most discriminating features between spam and ham, their analysis revealed three-way-handshake, time-to-live, idle time between packets and variance of inter-packet delay. Performance-wise, they showed that transport-layer features are stable over time and can classify spam with 85–92% accuracy.

Esquivel et al. [42] suggested leveraging transport-layer characteristics to defend against spam at the router level using a signature-based defense mechanism. For this to be accomplished, the mechanism has to be lightweight so that it does not impose overhead on the router; that is, the signatures have to be stateless and require a small amount of memory. TCP fingerprints were proposed as signatures because they are lightweight, can be computed on a single TCP SYN packet, and are very few in number, so they can be stored without much overhead incurred on the router.

They experimented on two live e-mail data sets that included both spam and legitimate e-mail messages. They used ***pOf*** [43] as the tool to extract signatures from a packet capture. They discovered common signatures across both data sets for spam e-mails; however, in the case of legitimate e-mail messages, no common signatures were revealed. Noteworthy was the fact that many of the top signatures used by hosts to sent legitimate messages are also used by hosts to send spam, which is evidence that they have changed some of the OS configurations. Furthermore, they observed that the spam signatures were stable over a period of several months for both locations. As far as performance, analysis showed that router-level filtering with TCP fingerprinting can filter 28%–59% of spam messages with a 0.05% false-positive rate.

Another approach on traffic characterization was proposed by Schatzmann et al. [44]. They focused on the network-level characteristics of spammers, but from the perspective of an AS or ISP. Their idea is based on the assumption that a large number of e-mail servers perform some level of pre-filtering (e.g., blacklisting). This knowledge however, remains local at the server and, depending on the server configuration or the

policy that is in effect, each server would perform differently. If we had access to that knowledge, we could analyze it and use it to improve the overall performance of the servers.

Schatzmann et al. showed that this *local* knowledge of pre-filtering decisions can be collected using flow-size information like bytes per flow, packets per flow, or average bytes per packet instead of examining the server logs. They gathered data within a three-month period from border routers of a major ISP serving more than thirty universities and government institutions. Their analysis showed that 95.64% of the sessions that failed had flows with less than 322 bytes, 96.99% of the sessions that were rejected had flows from 322 to 1559 bytes (corresponding to the SMTP envelope), and 97.16% of the sessions that were accepted had flows of greater than 1559 bytes. So just by byte count we can estimate the filtering decisions of mail servers.

They further validated their claim on a network-wide scale with fifty active mail servers that used blacklisting and whitelisting. The results showed that the traffic rejected by blacklisting had flow sizes between 322–1559 bytes, which concur with the above findings, and more than 90% of the accepted SMTP sessions had flow sizes greater than 1559 bytes. Leveraging this knowledge, they further proposed a reputation-rating system of e-mail senders. This intuition is based on the fact that when a server rejects in a consistent manner, it implicitly applies a rating on the specific client. These ratings can be used to build a collaborative-rating system, where the system would recommend acceptance or rejection of an SMTP session based on the collective behavior of all the servers.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ENVIRONMENT-SYSTEM OVERVIEW

This chapter describes the architectural approach we followed for our system and environment. We evaluated our system in both a virtual test bed and a live-test environment. A virtual test bed provides insights about the behavior of a system and allows for more controllable conditions, allowing us to reach more reliable and reproducible results [2]. Our goal was to evaluate our system under high-traffic-rate conditions and measure characteristics such as throughput and system load. This is especially appropriate when deploying the system in a resource-constrained environment such as a router.

Live testing, on the other hand, is important because it reveals how the system interacts with possibly unknown features of the external environment [2]. We deployed our system in a live environment from January 25, 2010 to March 2, 2011 and collected a trace of 5,926 e-mail messages. Section A describes the architecture of our virtual test-bed environment and Section B discusses the design of our real-time, abusive-network, traffic-detection system.

A. VIRTUAL-ENVIRONMENT ARCHITECTURE

An overview of our virtual environment is shown in Figure 2. It consists of three building blocks: the client side, server side, and network emulator. The client side generates the required SMTP [4] traffic, which is then received, analyzed, and classified on the server side. The role of the network emulator is to simulate congestion, in the form of longer delay, delay variance, retransmissions, etc., that large volumes of spam traffic will cause on the link.

1. Server Side

The server side consists of two virtual machines: one acting as the DNS server and the other as the mail server (MTA). For our DNS server, we used BIND [45], and for our mail server, Postfix [46].

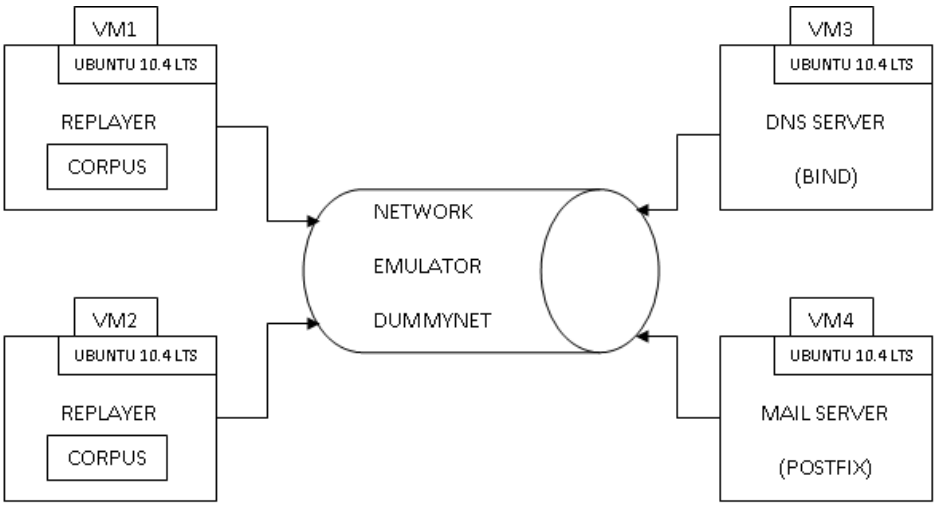

Figure 2. Virtual-Environment Architecture

Furthermore, we installed SpamAssassin [47] and SpamFlow [3] in the virtual machine that hosts the mail server, in order to achieve real-time, traffic-analysis functionality. More specifically, we integrated SpamFlow into SpamAssassin by developing a Perl plug-in so that SpamFlow could analyze SMTP packets, build the flow for each message, and extract the corresponding TCP features in real time.

Further, we have a libpcap [48] process that is running in promiscuous mode to assist SpamFlow in accomplishing its tasks. It collects all passing traffic from the interface it is listening on and writes them to a file, which is rotated at a specific rate to avoid large files and still not miss packets that belong to a message we haven't processed yet. Note that in the future, we plan to more tightly integrate the promiscuous packet capture with SpamFlow by storing flow features in memory as possible to avoid file system performance overhead.

For classification, we developed a classifier in Python using Orange [49], a machine-learning library for Python. See Section B for the implementation details of the classifier and the approach we selected to establish a communication path between SpamFlow and the classifier.

2. Client Side

The client side also uses two virtual machines that accommodate the task of generating the appropriate SMTP traffic. Each client consists of an e-mail replayer and the 2005 TREC public spam corpus [50] containing 92,187 messages, of which 52,788 are spam and 39,399 are legitimate. The corpus consists of an index file and a directory structure with the messages. The index file contains the path and label—spam or ham—for each message, which we use to establish ground truth. One issue that we had to address during our design was the limited TCP ephemeral-port range that the operating system enforces with regard to the volume of our corpus, as we discuss further below.

For the purpose of generating our SMTP traffic, we developed in Python [51] an e-mail replayer which performs the following tasks:

- takes as input each message from a corpus, extracts the headers, and adds as recipient a valid user of our virtual-network domain
- adds another helper header that contains the label of the message in such a way as to not trigger SpamAssassin and enables us to establish the ground truth during our analysis process
- establishes an SMTP session with our mail server
- sets the type of service (*tos*) field in the IP header of each message to some value, depending on its class. Thus spam and legitimate messages have different *tos* values, which allows us to redirect them through different paths in our network emulator
- finally, the replayer transmits the message

As mentioned above, the operating system limits the range of ephemeral[3] ports a host can use. In our case, the range of available ephemeral ports is from 32,768 to 61,000, which allows us to establish 28,232 unique TCP connections. The total number of messages we want to transmit, however is 92,187, many more than the available connections. This is a problem because we use the IP:Port tuple to identify the message,

[3] Ephemeral ports are temporary ports assigned by a machine's IP stack, and are assigned from a designated range of ports for this purpose.

build the flow from the message packets that correspond to the given IP:Port tuple, and extract the features. As a solution, we used two virtual machines and manually bound the interface to a port using our own ephemeral-port range. Applying these two approaches, every message was mapped to a unique 4-tuple (server IP, server port, host IP, host port), which allowed us to uniquely identify each message on the server side and extract its corresponding flow features. We could have adopted another approach and used the message identifier instead, which is unique for every message. This approach, however, would require making a deep packet inspection to retrieve the message identifier, which implies more computational time, and we would lose the lightweight principal from our system.

3. Network Emulator

Emulators are tools that generate appropriate network-environment characteristics to allow for protocol or application evaluation. In our case, our goal is to reproduce the TCP characteristics that spam TCP traffic exhibits, such as TCP timeouts, retransmissions, resets, and highly variable roundtrip time (RTT) estimates [3]. For our evaluation, we selected Dummynet [52], a publicly available tool that allows packets to pass through virtual network links to introduce delay, loss, bandwidth constraints, queuing constraints, etc.

Dummynet [2] comprises two main components: an emulation engine and a packet classifier. The emulation engine (Figure 3) or *pipe* as we will call it, consists of a finite-size queue, a scheduler, and a communication link with fixed bandwidth and programmable propagation delay. We can build our network environment by configuring the main parameters: bandwidth, queue size, queuing discipline, and propagation delay. Traffic is passed to the pipe using the packet classifier, *ipfw,* which matches packets according to a predefined rule set and applies appropriate actions.

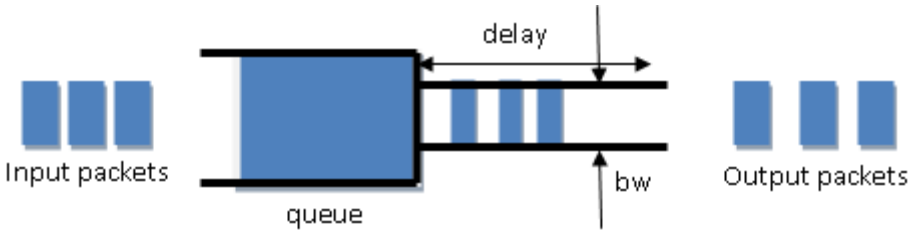

Figure 3. Dummynet [From [2]]

Once a packet is inserted into the pipe, it is queued and drained at a rate corresponding to the link's bandwidth B. The next stage for the packet is the communication link, where it stays for a time t_D equal to the propagation delay of the link. The packet is reinjected into the network stack after time t_D expires. As a result of this process, the pipe will delay each packet i by a time $T_i = (l_i + Q_i)/B + t_D$, where l_i is the length of the packet, Q_i is the queue size, and B and t_D are the bandwidth and propagation delay of the link, respectively [2].

As mentioned above, Dummynet introduces a fixed amount of propagation delay into the link, which induces difficulties in achieving a variable RTT, as would be present in a congested environment. To address this problem, we modified Dummynet to provide random delays based on a normal distribution with mean μ and standard deviation σ. More specifically, we set up Dummynet to introduce a mean delay of 150 ms with 50-ms standard deviation for spam traffic that originates from the replayer and is destined for the mail server, and a 40-ms mean delay with 25- ms standard deviation for legitimate traffic in both directions. We introduced delay in legitimate traffic in order to avoid overfitting our model.

To emulate timeouts, retransmissions, and resets, we applied a random-packet-drop policy on the pipe. While we recognize that our modifications to Dummynet only partially emulate a congested network (for example, loss events are independent—an assumption that does not hold true in a real queue), our goal in the emulation

environment was to enable testing. Specifically, as mentioned above, the environment provides a means to emulate high-rate traffic and evaluate performance, throughput, system load, etc. on representative traffic.

B. SYSTEM DESIGN

An overview of our real-time system is shown in Figure 4. It comprises four main components: SpamAssassin, SpamFlow Analysis Engine, SpamFlow Plug-in, and the SpamFlow Classification Engine. We refer to SpamFlow Analysis Engine, SpamFlow Plug-in, and SpamFlow Classification Engine as *spamflow*, *plugin*, and *classifier*, respectively. Furthermore, we have a separate process running in promiscuous mode, which captures every packet of the SMTP session using libcap and stores it to disk.

Every message received by the mail server is processed by SpamAssassin and then piped to *plugin*, where we extract the identification tuple (host IP address, host port number) from the message and then pass it to *spamflow* for feature extraction. Thereafter, *plugin* is responsible for communicating with *classifier* for the classification task. We describe each component in more detail in the following subsections.

1. SpamAssassin

SpamAssassin is an open-source, rule-based, content filter. Each rule is assigned a score using a genetic algorithm. All scores are then aggregated to produce an overall score for each message. The classification process involves comparing the overall score with a user-defined threshold (which defaults to a value that maximized performance on a broadly representative training sample during the genetic-algorithm stage). If the score is above the threshold, then the message is classified as spam; otherwise, as legitimate. Moreover, using a modular architecture, SpamAssassin can be extended to include other filtering techniques, such as real-time blackhole lists (RBLs), whitelists, collaborative filtering, learning-based techniques (e.g., naïve Bayes), and others.

Furthermore, SpamAssassin features a threshold-based mode in which new exemplar emails trigger an automatic retraining process. While the SpamAssassin documentation refers to this as “auto-learning,” in the machine learning and spam-

filtering communities, this is typically called online or iterative learning. The primary difference is that in advanced iterative learning approaches the classification model is modified to account for the new emails; whereas in auto-learning, the entire model is recreated. More specifically, SpamAssassin selects messages that achieve proper threshold values, rebuilds the model of the built-in naïve Bayes classifier, and classifies subsequent messages with the newly updated model. A message is selected as spam if the score that it receives is greater than 12 points and as non-spam if the score is less than 0.1 points. We discuss the results of applying this technique to our system in Chapter IV, Section C.

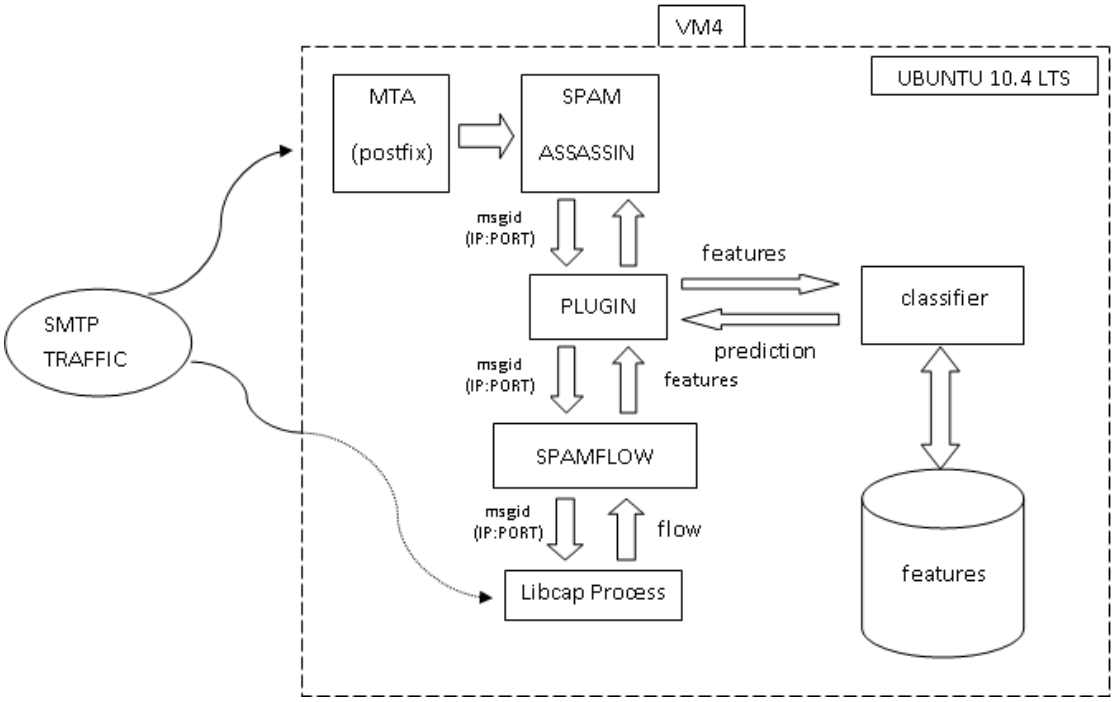

Figure 4. System Architecture

We used SpamAssassin with the default configuration of rules, but we disabled all network tests (lookups in blackhole lists (RBL), collaborative filtering with Ryzor [30], Pyzor [53], and DCC [33]), because our virtual environment was insulated from the outer world. Also, we disabled rules comparing the date on the message header with the date

the message was received. All messages in our corpus, whether legitimate or spam, have out-of-date dates in the headers, which implies that triggering those rules makes no contribution to the overall scoring process (and might artificially affect classification performance negatively).

2. SpamFlow

Spamflow serves as our network analyzer. It accepts as input a libpcap [54] trace, builds corresponding flows, and extracts TCP features for each flow. We modified *spamflow* for our purposes to extract TCP features for a given message identified by the (host IP address, host port number) tuple. To accomplish the above tasks, *spamflow* needs to communicate with a separate process that is responsible for capturing TCP packets in promiscuous mode and storing them to disk. *Spamflow* retrieves the packets from the disk associated with the given (IP address, TCP port number) tuple and extracts their flow features. This solution, however, is not the most effective since it involves file operations. A better solution that also preserves the lightweight character of our system is for all input/output operations of both the libpcap process and *spamflow* to be executed in memory. We leave this as future work and in the meantime could use a RAM disk to emulate this behavior. We modified our mail server to add to the header of each e-mail the (IP address, TCP port number) identification tuple of the remote mail-transport agent (MTA) sending the mail.

3. SpamFlow Plugin

Spamflow cannot operate as a standalone application for real-time traffic analysis; therefore, we had to integrate it with an existing one. We selected SpamAssassin because it is open source, widely used (the commercial Barracuda [55] network appliance is based on SpamAssassin), and employs a modular architecture that allows extensions through plug-ins. SpamAssassin is written in Perl [56]. We developed, using Perl, a module that integrates *spamflow* into SpamAssassin and allows it to operate in a real-time fashion: as e-mail messages are routed through *spamflow*, they are classified using a previously

learned model of transport features and given a score. This score, in combination with the scores from other rules, provides a final message disposition.

Plugin acts as the controller of the system and binds the traffic-analysis engine and the classifier together. This module performs two main tasks that are related to *spamflow* and *classifier*. The first task is to provide *spamflow* with the 2-tuple identifier of the current message under inspection and receive in return the features that correspond to the given message identifier. Once *plugin* obtains the features, the second task involves classification: passing the features, via an appropriate protocol, over to the classifier and retrieving the corresponding classification. In Figure 5, we see an example of a message headers where the *plugin* has attached the features that *spamflow* has extracted.

```
From Josephine@rsi.com Tue Feb 01 23:21:58 2011
Return-Path: <Josephine@rsi.com>
X-Spam-Checker-Version: SpamAssassin 3.3.1 (2010-03-16) on
ralph.rbeverly.net
X-Spam-Level: **
X-Spam-Status: No, score=2.9 required=5.0
tests=BAYES_40,HTML_MESSAGE,SPAMFLOW,
          UNPARSEABLE_RELAY autolearn=no version=3.3.1
X-Spam-Spamflow-Tag: the features are
3792891725:37689,12,10,0,0,0,0,1,1,0,53248,34.464852,0.162818,120.44
1156,148.297699,51.891697,5840,48,1,64
Delivered-To: rbeverly@ralph.rbeverly.net
Received: (qmail 30923 invoked by alias); 1 Feb 2011
23:21:58 -0000
Delivered-To: jobs@eactivenetworks.com
Received: (qmail 30920 invoked from network); 1 Feb 2011
23:21:57 -0000
Received: from cm-static-18-226.telekabel.ba
(77.239.18.226:37689) by ralph.rbeverly.net with SMTP; 1 Feb
2011 23:21:57 -0000
Received: from vdhvjcvivjvbwyhxnscvfwq (192.168.1.185) by
bluebellgroup.com (77.239.18.226) with Microsoft SMTP Server
id 8.0.685.24; Wed, 2 Feb 2011 00:20:48 +0100
Message-ID: <4D489025.504060@etisbew.com>
Date: Wed, 2 Feb 2011 00:20:48 +0100
From: Essie <Essie@hermes.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.12)
```

Figure 5. Message Headers with *spamflow* features

To accomplish this task, we used XML-RPC [57] as the communication protocol between the two ends. XML-RPC is a simple protocol that allows procedures running in different applications or machines to communicate with each other. They exchange XML-formatted data [58] using the HTTP [59] protocol. Specifically, the client uses the HTTP-POST request to pass data to the server; the server in return sends an HTTP response. In our implementation, we registered *classifier* with a *classify* procedure that takes as input the features. So *plugin* sends the HTTP-POST request with the name of the procedure to call, *classify*, along with the features, as coma separated values forming a string, and receives via HTTP response the classification label from each classifier.

We chose XML-RPC because it is simple. It allows *classifier* to potentially operate on a different machine from *spamflow*, which in the future could allow the XML-RPC *classifier* to serve many *spamflow* instances in a multithreaded fashion and distribute load; it uses XML, which has been established as a standard format across many platforms and applications; and finally, both Perl and Python provide an API for the XML-RPC protocol. The latter has to do with the fact that we implemented *classifier* using Python, as we will discuss in the next subsection.

4. SpamFlow Classification Engine

As mentioned above, we have implemented *classifier* using Python and the Orange machine-learning package. Orange is a simple-to-use package with many features, such as a variety of machine-learning algorithms, a statistical module that allows different evaluation techniques, and visualization widgets.

Our *classifier* implementation comprises three machine-learning algorithms: naïve Bayes, decision trees (C4.5), and support-vector machines (SVM). We selected three algorithms because we wanted to examine if the classification performance of our system is a function of the classification method, and these algorithms are known to provide good performance. All three algorithms are invoked to provide a classification for the requested flow features.

IV. EXPERIMENTAL METHODOLOGY AND RESULTS

In this chapter, we describe the experiments performed on our system using the infrastructure and architecture described in Chapter III. We then discuss performance results and the implications as observed in both the synthetic laboratory environment and in the live network test.

A. EXPERIMENTS

We evaluated both the system as a whole and *classifier* by itself, with three different experiments. The first two were performed on the virtual test-bed environment and the third on a server in live testing. For both experiments on the virtual test bed, we tested our system against high-rate and high-volume traffic. Our volume consisted of approximately 70,000 e-mail messages that we replayed using both of our clients. While the combination of the replayer application and the simulated network delay limited our testing throughput to approximately 110 msgs/min, we show later in this chapter that our classification engine is capable of processing 78,000 msgs/min. The load to which we subject the system is sufficiently high to simulate a realistic environment, exercise our algorithms, and measure performance with reliability.

As stated in Chapter III, our other goal was to evaluate our system classification performance. Even though the conditions that our virtual test bed produced are, naturally, not perfectly representative of a live environment, we can extract valuable information on how the classifiers react and perform on artificial features. The test bed gives us the ability to measure different performance metrics such as throughput, training times, and system load. We established a reference of comparison for evaluation purposes by using SpamAssassin in two configurations. The first configuration included only the local tests, which perform checks on the message body and headers, whereas in the second phase we allowed SpamAssassin to utilize its own built-in naïve Bayes classifier. As previously stated, we did not use any rules requiring network access, e.g., real-time blacklists, because our environment had no access to the Internet.

To accommodate our experimental needs, we divided our corpus into two sets. The testing set consisted of 70,000 messages, and the remaining 22,176 messages were the training set. For each experiment, we used four different training sets, consisting of 10, 100, 1,000 and 10,000 training examples. Each training set was then populated using sampling without replacement, as we did not want to select the same message twice, ensuring an equal composition in the number of spam and legitimate e-mail messages. For each training-set size, we perform tenfold cross-validation to ensure generality and reliable results. Our goal was to evaluate our system with increasing sizes of training examples and repeat the evaluation process for each set ten times, in order to compensate for any variations that would be a result of the sampling procedure, as well as to affirm that the classifiers would perform better as the training set size increased.

The metrics that we used for our evaluation are the standard classification performance measures of accuracy, precision, and recall. Precision measures classification specificity [60], which is defined as the proportion of the positive classifications that are correct; this measure penalizes any incorrect positive classification (false positives) but does not penalize any loss of positive classifications. Recall measures classification coverage [60], defined as the proportion of the set of positive cases that the system correctly classifies, and therefore penalizes false negatives but not false positives. Accuracy measures the proportion of classifications that are correct, and thus gives equal weight to both false positives and false negatives. The formulas that we used to compute the aforementioned metrics are the following:

$$precision = \frac{t_p}{t_p + f_p}$$

$$recall = \frac{t_p}{t_p + f_n}$$

$$accuracy = \frac{t_p + t_n}{\left(t_p + t_n\right) + \left(f_p + f_n\right)}$$

Finally, we need to mention the SVM configuration that we used in our evaluation, since it is tunable. The SVM implementation of Orange provides two classification types, C-SVC (support-vector classification) and Nu-SVC, with each

adjusting different parameters to calibrate their performance. For our experiments, we selected the C-SVC type, because Nu-SVC, which was the default type in Orange, was producing errors, and C-SVC was recommended instead. As kernel type, we used the Gaussian kernel or radial-basis function (RBF), because it is proposed in [61] as a good start, and we selected parameter C[4] = 5. We leave as future work examination of the performance of our system on different kernel types and parameters.

B. RESULTS

1. Test-Bed Evaluation

Our goal for this first experiment was to evaluate our system in a high-volume-and-rate traffic environment and examine the system and classification performance. In the first subsection, we discuss the classification performance relative to SpamAssassin as a reference point, and in the second subsection, present the system performance with respect to throughput, classification training times, and system load.

a. Classification Performance

The results of our system evaluation (first experiment, lab environment) with respect to accuracy, precision, and recall are shown in the following figures. The values in the x-axis are in logarithmic scale (base 10) and represent 10, 100, 1,000 and 10,000 training examples.

Figure 6 shows accuracy, where we can observe that all three classifiers (naïve Bayes, C4.5 decision trees, and SVM) behaved as expected. As the number of training examples increased, the resulting performance increases. With large training-set sizes, all of the classifiers achieved greater than 98% accuracy on our test data. C4.5 had the best performance of all, achieving 99% accuracy; but in general we cannot detect any significant performance difference among the different classifiers.

[4] C defines the penalty parameter of error term and is always greater than zero.

On the other hand, SpamAssassin, using only word-token features, produced only 99 true positives—a 40% accuracy, which is quite a significant difference, even in an artificial environment. These results in the laboratory suggest the significant potential effectiveness of TCP behavioral filtering versus content filtering. The analytical results on accuracy for the three classifiers and for each sample are shown in Table 1.

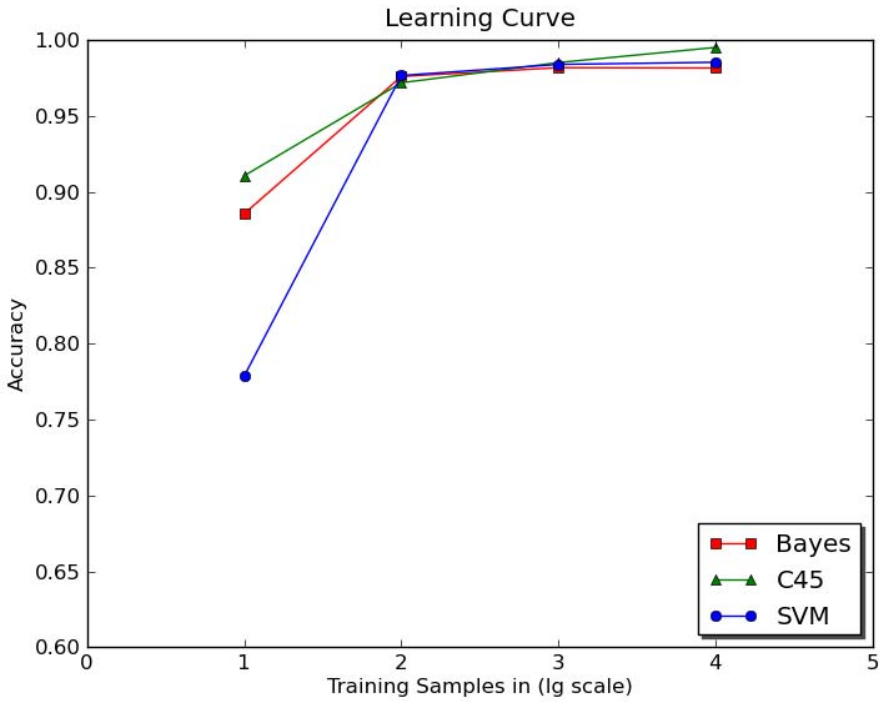

Figure 6. Test-Bed Evaluation: Accuracy

	Training Samples			
Classifier	10	100	1000	10000
Bayes	0.914	0.972	0.981	0.981
C45	0.900	0.966	0.987	0.996
SVM	0.812	0.977	0.983	0.978

Table 1. Test-Bed Evaluation: Accuracy

Precision and recall are depicted in Figure 7 and 8 respectively, and the analytical results for all classifiers in Tables 2 and 3. We observe again that all classifiers reach high rates of precision and recall. C4.5 has the most stable behavior, achieving a 99% rate both in precision and recall, followed by naïve Bayes and then SVM. By contrast, SpamAssassin without naïve Bayes performed well, precision-wise, with zero false positives, but had a very low recall rate of 0.02%, because there were a great number of false negatives.

Noteworthy is the fact that all three classifiers reached the maximum performance rate after the second training sample, which is probably due to the synthetic traffic that we generated. We therefore suggest, as future work, the creation of a more representative synthetic traffic in the laboratory environment. Nevertheless, our results suggest that we may expect an analogous behavior in a live environment, achieving a good performance with a small number of training examples. The confusion matrices for each classifier are shown in Tables 4, 5, and 6.

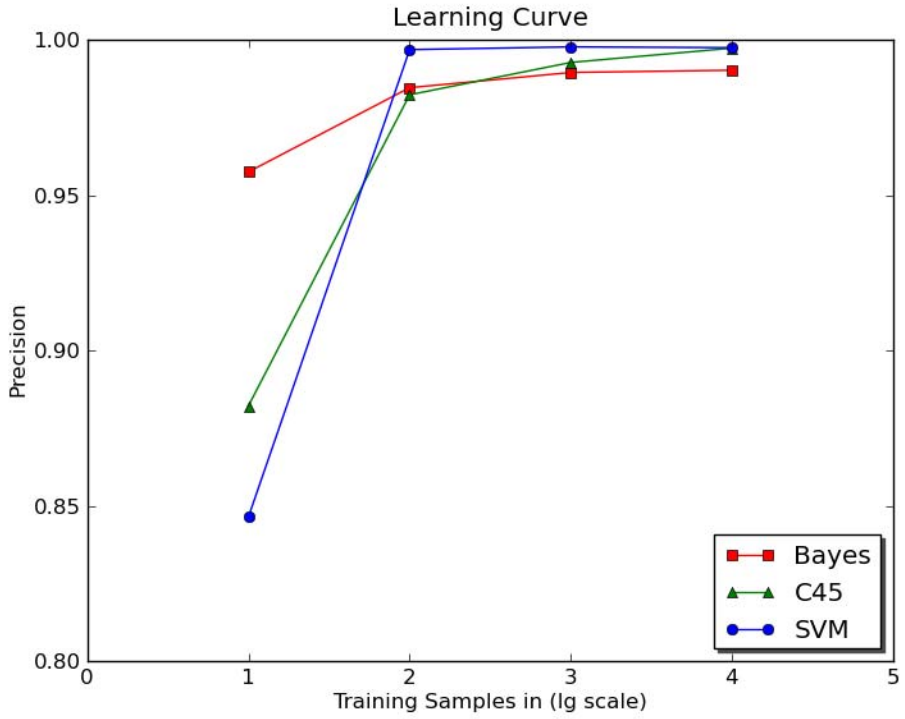

Figure 7. Test-Bed Evaluation: Precision

	Training Samples			
Classifiers	10	100	1000	10000
Bayes	0.938	0.983	0.989	0.99
C45	0.904	0.972	0.994	0.998
SVM	0.941	0.997	0.997	0.995

Table 2. Test-Bed Evaluation: Precision

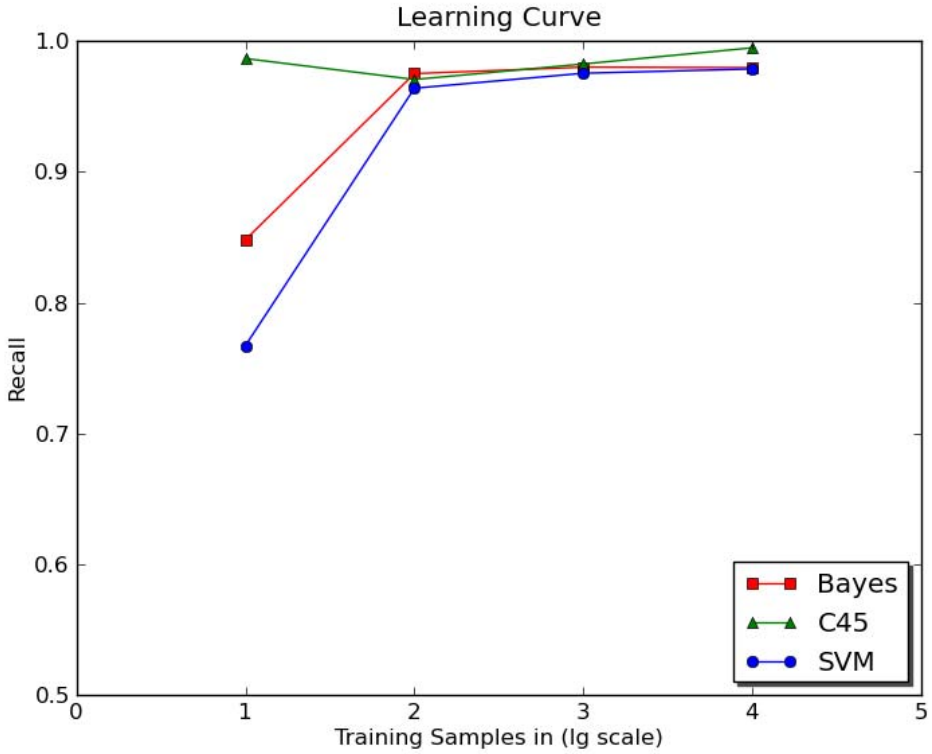

Figure 8. Test-Bed Evaluation: Recall

	Training Samples			
Classifiers	10	100	1000	10000
Bayes	0.921	0.970	0.979	0.979
C45	0.939	0.971	0.984	0.995
SVM	0.738	0.964	0.975	0.970

Table 3. Test-Bed Evaluation: Recall

Sample	TP	FN	TN	FN
10	38414	3275	25559	2739
100	40399	1245	27557	696
1000	40342	852	27361	442
10000	35940	754	22944	360

Table 4. Test-Bed Evaluation: Naïve Bayes Confusion Matrix

Sample	TP	FN	TN	FN
10	39127	2562	23889	4409
100	40450	1194	27081	1172
1000	40536	658	27548	255
10000	36521	172	23218	85

Table 5. Test-Bed Evaluation: C45 Confusion Matrix

Sample	TP	FN	TN	FN
10	30782	10908	26055	2243
100	40141	1503	28152	102
1000	40162	1032	27694	109
10000	35583	1111	23115	188

Table 6. Test-Bed Evaluation: SVM Confusion Matrix

b. Throughput—Load

Table 7 shows how the three classifiers performed with respect to training times and classification throughput. Examining the results, we observe that naïve Bayes provides the higher throughput among the three classifiers, and this conforms to the fact that its decision rule is much simpler than the other two, whereas C4.5 has the lowest training time. SVM, on the other hand, achieves the lowest throughput and the largest training time, due to the more complex decision model. The significant takeaway from these measurements is that, taking into account the relative independence of our system from the classification method, we can select the classification model that fit our needs. For example, the low training time of C4.5 makes it a good candidate when we need to retrain often and want to minimize idle times.

	Training Times (msec) across samples				Throughput (msgs/sec)
	10	100	1000	10000	
Bayes	0.884	15.016	105.453	104.843	1300
C4.5	0.151	0.964	16.017	29.785	1100
SVM	0.721	12.691	224.250	260.018	700

Table 7. System Performance

The classification engine's CPU utilization, using all three classifiers at the same time, for the duration of our experiment was 0.1%, as we observe on Table 8, which shows that our system requires low system resources in order to operate.

	Time (sec)
User	6.52
System	86.04
Elapsed	86263
CPU	0.1%

Table 8. Classification Engine CPU utilization

2. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode

In the second experiment, we explore if we can improve the built-in naïve Bayes classification engine of SpamAssassin using the threshold-based "auto-learning" operational mode that we described in Chapter III, Section B.

This experiment allows us to assess the strength of TCP features against the content features. Accuracy, precision and recall of all three classifiers are presented in Figures 9, 10, and 11, the analytical results are presented in Tables 9, 10, and 11. Again we observe as in the first experiment that C4.5 and Bayes achieve high performance across all metrics with rates above 95%. SVM, however shows a deviation compared to the results of the first experiment, which is probably due to the fact that we have not tried different SVM kernel types and parameters.

SpamAssassin, on the other hand, does not show any improvement using the naïve Bayes classifier in "auto-learning" mode, compared to the previous experiment. As is shown in Table 12, accuracy and recall remain at the same low levels of 40% and 0.2% respectively. Nevertheless, we stress that SpamAssassin achieves zero false positives, that is 100% precision rate, which implies its conservative character, and that a combination of traffic and content filtering would produce a good line of defense against spam. The confusion matrices of all three classifiers are shown in Tables 13, 14, and 15.

	Training Samples			
Classifier	10	100	1000	10000
Bayes	0.817	0.961	0.970	0.969
C45	0.812	0.963	0.983	0.993
SVM	0.599	0.968	0.962	0.870

Table 9. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Accuracy

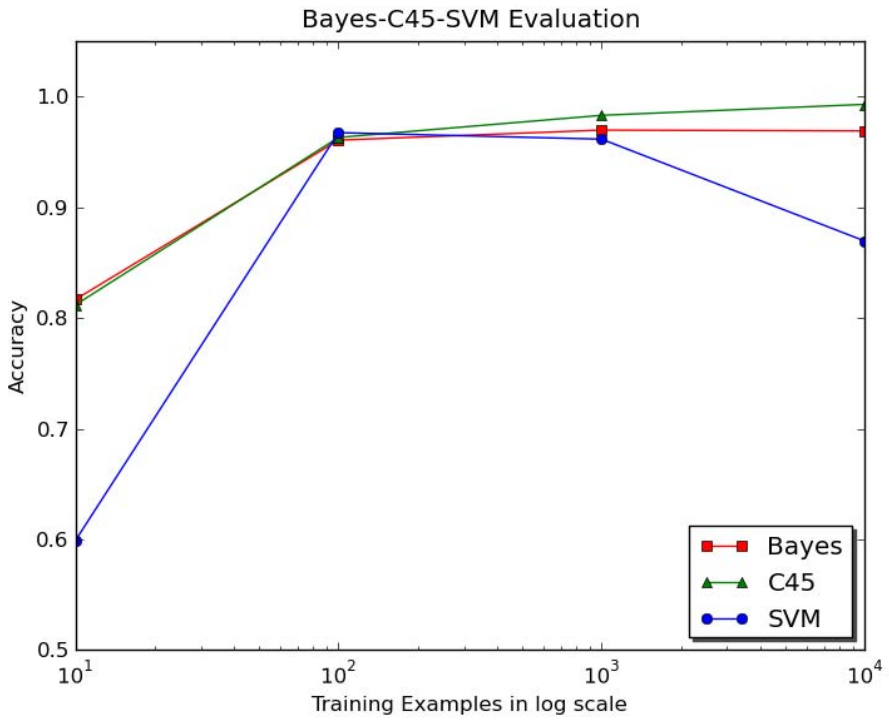

Figure 9. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Accuracy

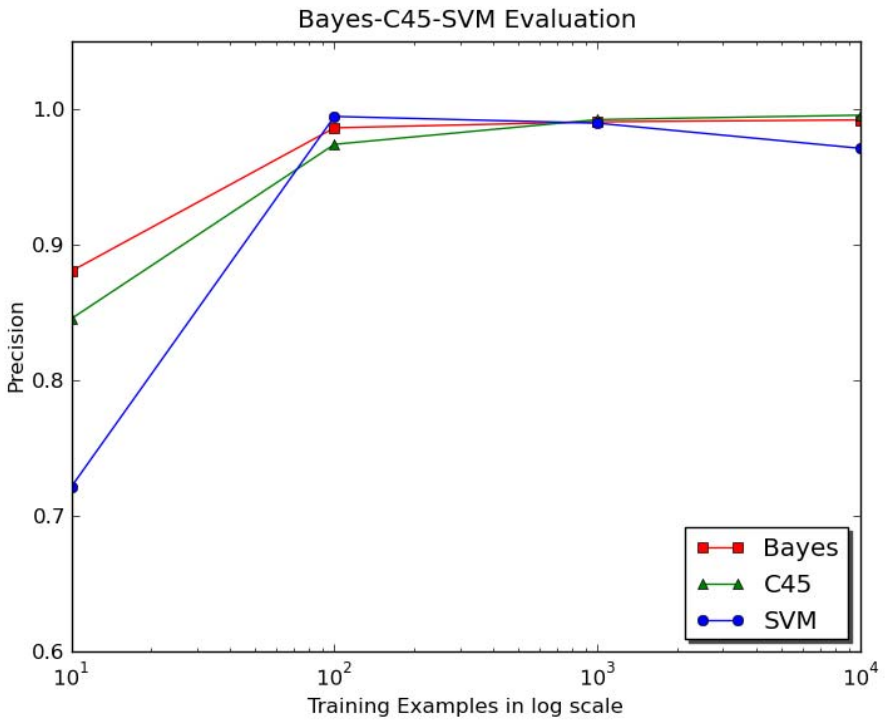

Figure 10. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Precision

	Training Samples			
Classifier	10	100	1000	10000
Bayes	0.881	0.986	0.991	0.992
C45	0.846	0.974	0.993	0.996
SVM	0.722	0.995	0.990	0.971

Table 10. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Precision

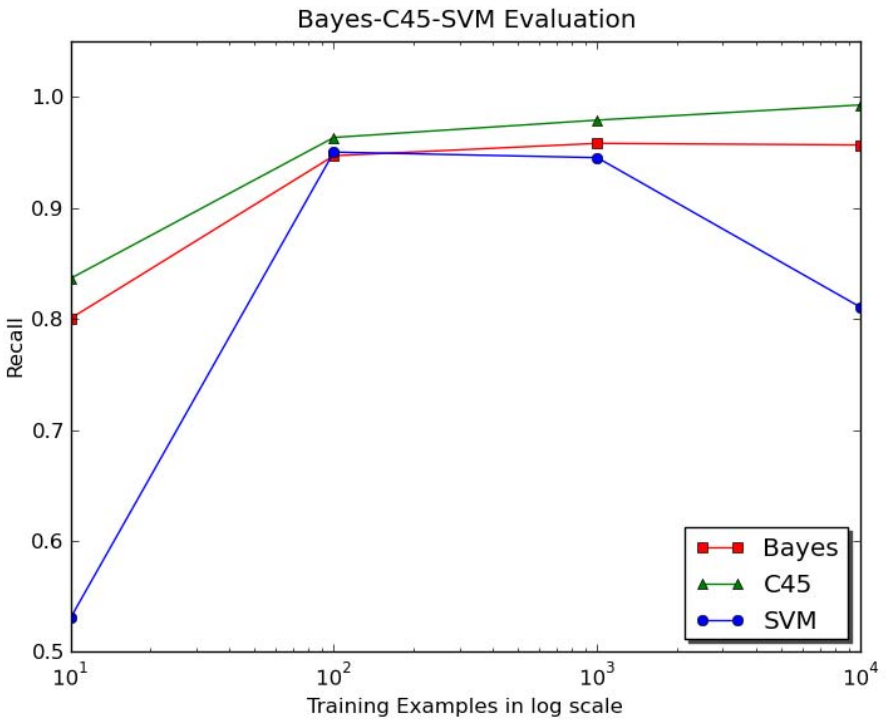

Figure 11. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Recall

	Training Samples			
Classifier	10	100	1000	10000
Bayes	0.800	0.947	0.958	0.957
C45	0.836	0.964	0.979	0.993
SVM	0.530	0.950	0.945	0.811

Table 11. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Recall

SpamAssassin			
TP	99	Accuracy	0.4
FN	41595	Precision	1
TN	28304	Recall	0.002
FP	0		

Table 12. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: SpamAssassin Results

	TP	FN	TN	FP
10	32910	8209	23525	4457
100	38902	2171	27402	535
1000	38929	1695	27129	358
10000	34566	1557	22718	269

Table 13. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: Naïve-Bayes Confusion Matrix

	TP	FN	TN	FP
10	34394	6724	21703	6279
100	39579	1494	26891	1046
1000	39780	843	27187	300
10000	35869	254	22834	153

Table 14. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: C45 Confusion Matrix

	TP	FN	TN	FP
10	21804	19314	19572	8410
100	39036	2037	27734	203
1000	38406	2217	27094	393
10000	29279	6844	22121	866

Table 15. Test-Bed Evaluation Using SpamAssassin in Auto-Learning Mode: SVM Confusion Matrix

3. Live Testing

In this section, we discuss the results of our live testing on the rbeverly.net MTA from January 25, 2011 to March 2, 2011, where we collected 6,026 e-mail messages, of which 5,610 were spam and 416 legitimate. Ground truth was first established via SpamAssassin. We then manually examined all the legitimate messages and relabeled those that were false negatives. We did not examine every spam message to ensure no false positives (due to the large volume of such messages); however, we did manually sample for spam to establish reasonable ground truth. Even though the volume of traffic captured is small and represents a small portion of the Internet traffic, the results with respect to accuracy, precision, and recall were strong.

The results are depicted in the following figures, where each figure presents the performance of the three classifiers on a specific metric. For the live testing corpus, we selected training sets of 8, 16, 32, 64, 128, 256, and 512 messages, following the same procedures (tenfold cross-validation, etc) as the simulated experiments and tested on the remaining messages. The values on the x-axis represent the number of samples in logarithmic scale (base 2); so for example, x-axis x=2 represents a sample of eight training examples, x=4 represents 16 training examples and so on, and each sample has an equal number of spam and legitimate e-mail messages. The values on the y-axis represent the percentage for each metric, and each curve maps to a classifier.

In Figure 12, we present the accuracy achieved as defined in Section A. First, we observe that all three classifiers behave well as the training size increases, with naïve Bayes achieving the higher accuracy, along with C4.5, but also having the smoothest curve. C4.5, also performs with the higher accuracy but has some fluctuations, considering the lower starting point and the knee at point six on the x-axis. These fluctuations, however, cannot be considered representative, due to account the small number of training examples. Finally, SVM has the lowest accuracy among the three but is more stable than C4.5. As we mentioned in Section A, SVM is very adjustable, so this performance may not be representative. The analytical results of all three classifiers and for each training sample size are shown on Table 16.

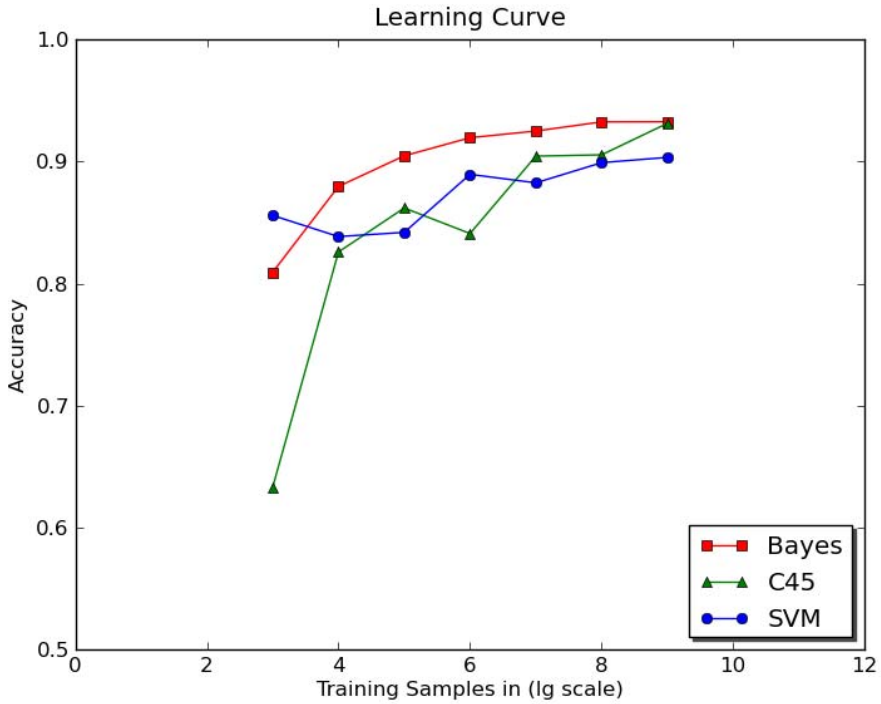

Figure 12. Live Testing: Accuracy

	Training Samples						
	8	16	32	64	128	256	512
Bayes	0.8	0.88	0.9	0.92	0.925	0.93	0.93
C4.5	0.63	0.83	0.86	0.84	0.9	0.91	0.93
SVM	0.85	0.84	0.84	0.89	0.88	0.9	0.9

Table 16. Live Testing: Accuracy

Another significant point is that the network behavior of spammers at the TCP level as it is captured by *spamflow* is so strong that accuracy is independent of the classification method that we want to follow. This is significant because we can select an algorithm that has low training time and good throughput, thus minimizing the overhead on systems that have constrained resources, or must deal with very high rates of abusive traffic.

Moreover, we emphasize the fact that even with as low as sixteen training examples, our system exhibits greater than 80% accuracy, which is important when training examples are few.

Precision is shown in Figure 13 and more detailed results in Table 17. Examining our results, we observe that our system has a relatively stable behavior across samples, with a rate greater than 97%. At the maximum training size, all three classifiers achieve the significant rate of 99%. This implies that our system exhibits a very small false positive rate, which is crucial for our users, since we do not want to misclassify and lose legitimate e-mails. Furthermore, we observe again that we can achieve high precision rates independent of the classification method. Combining this high precision along with high accuracy, we can conjecture that *spamflow* is a very promising system. Another significant result is that naïve Bayes achieves both the higher precision and the higher accuracy, which establishes it as a good candidate for our system.

Finally, in Figure 14 and Table 18, we present how the classifiers behaved with respect to recall. Recall appears to show the same trends as accuracy, and this behavior is

expected considering the high rates in precision. Accuracy takes into account both false positives and false negatives, and since precision is almost 100%, recall is the component that drives its behavior. So, as with accuracy, all classifiers achieved more than 90% recall and behaved in the same manner as accuracy.

	Training Samples						
	8	16	32	64	128	256	512
Bayes	0.992	0.991	0.992	0.99	0.992	0.993	0.996
C4.5	0.971	0.98	0.986	0.992	0.991	0.996	0.997
SVM	0.981	0.984	0.986	0.986	0.989	0.99	0.994

Table 17. Live Testing: Precision

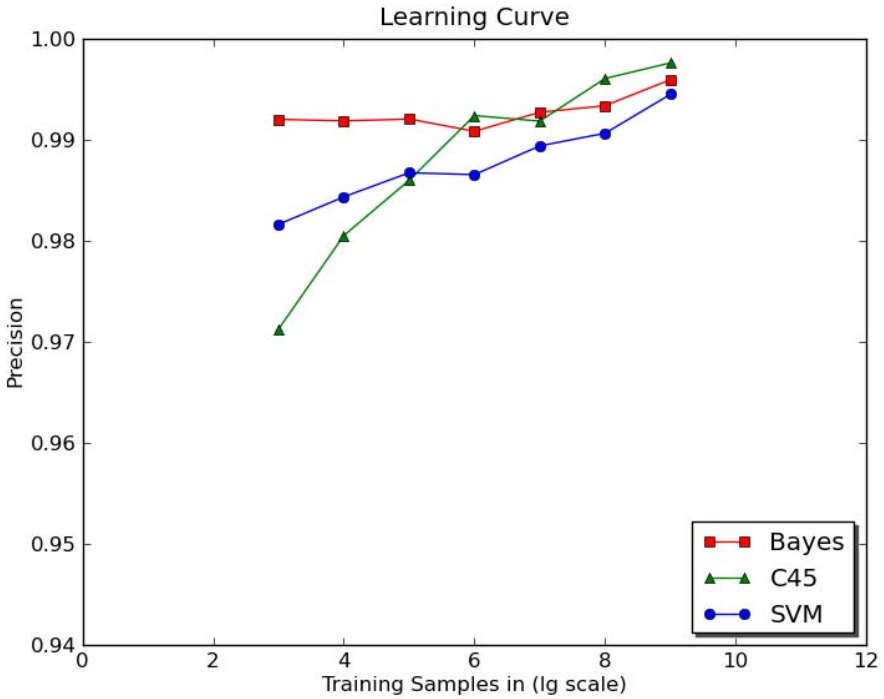

Figure 13. Live Testing: Precision

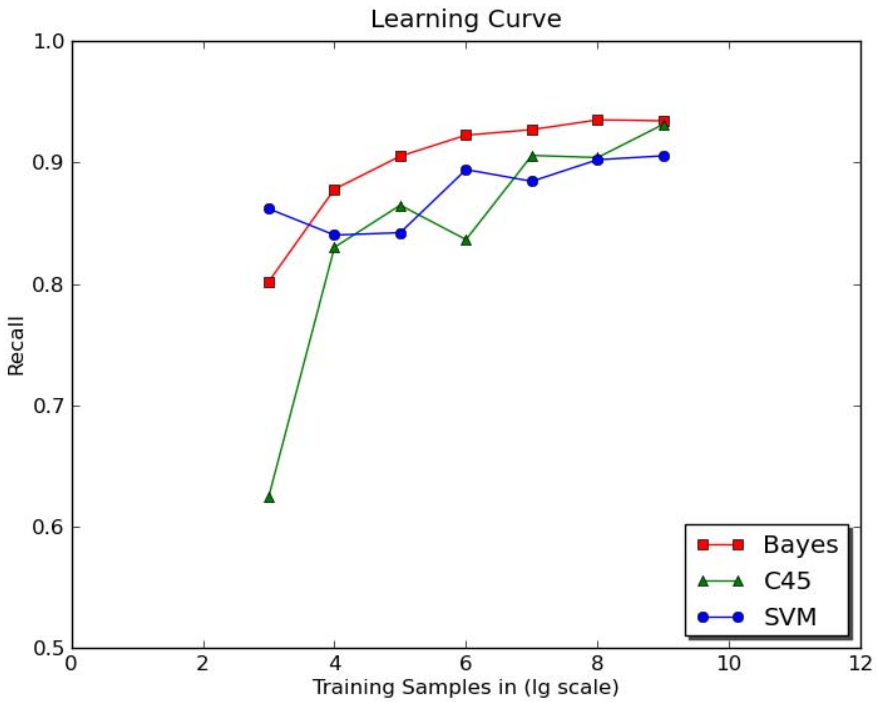

Figure 14. Live Testing: Recall

	Training Samples						
	8	16	32	64	128	256	512
Bayes	0.81	0.87	0.91	0.92	0.927	0.935	0.934
C4.5	0.62	0.83	0.86	0.83	0.906	0.904	0.931
SVM	0.86	0.84	0.84	0.89	0.88	0.902	0.905

Table 18. Live Testing: Recall

C. AUTO-LEARNING

In this section, we discuss some additional experiments we conducted on threshold-based auto-learning, as mentioned in Chapter III, Section B, and present their results. Auto-learning is the incremental process of building the classification model based on exemplar e-mail messages that achieve certain threshold values. In our case, we

use the flow features of e-mail messages otherwise classified via orthogonal methods as having very high or very low scores. More specifically, we explicitly retrain each classification model each time we observe a message with a particularly high score from the other SpamAssassin categories (rule- and Bayesian-word based) that meets our threshold criteria; i.e., having a score above or below our threshold. After retraining is complete, we evaluate our models on subsequent messages until we observe one or more messages with scores above or below our thresholds, at which point we stop and retrain the models. For this experiment we use our live corpus.

We set up two thresholds: one for spam messages and one for legitimate. The selection of the thresholds was based on the spam and ham score distributions. Spam-message scores follow a normal distribution, with mean $\mu = 16.31$ and standard deviation $\sigma = 7.73$, whereas scores of legitimate messages have mean $\mu = 1.3$ but are skewed to the left. Therefore, for the legitimate messages we selected a threshold t = 1, which proved effective as it allowed the classifiers to be trained on 267 messages out of the 416. For spam messages, we selected four thresholds to examine the trends of our classifiers. The first was the mean and the other three were one, two, and three standard deviations above the mean.

The results of the first run are shown Figures 15, 16, and 17, where we trained the classifiers with 2,685 spams and 267 legitimate messages, thus using ten times more spam than ham. The analytical results of the confusion matrix are presented in Table 19.

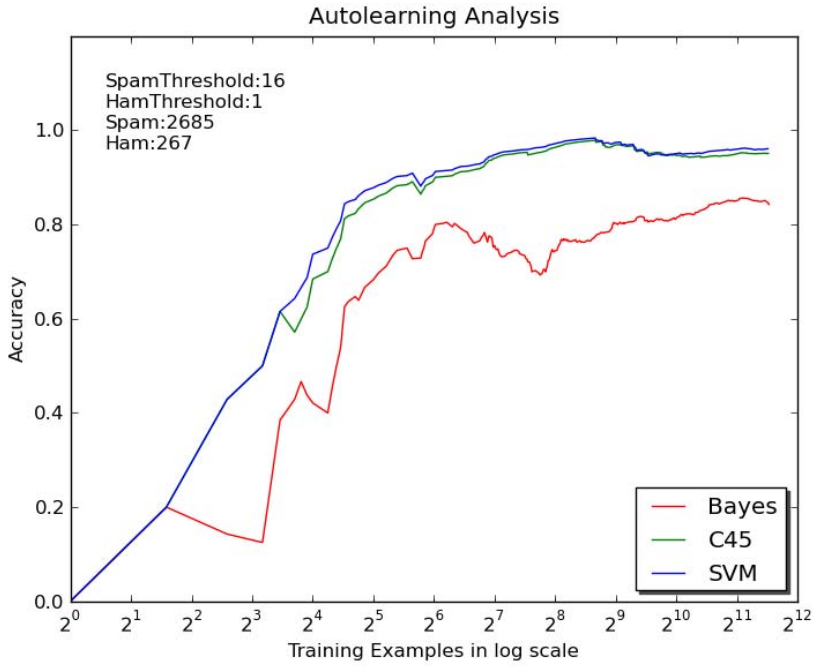

Figure 15. Auto-Learning (Threshold=16): Accuracy.

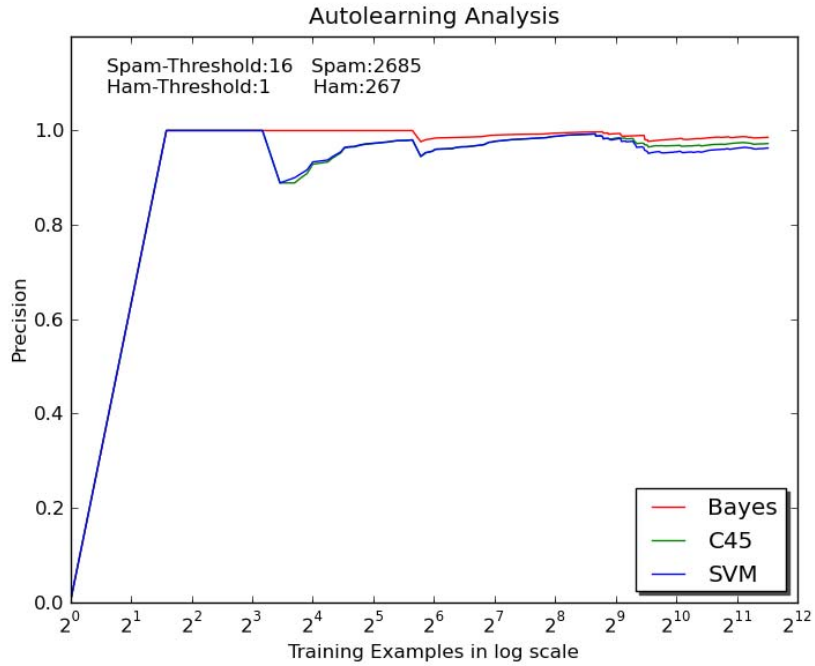

Figure 16. Auto-Learning (Threshold=16): Precision

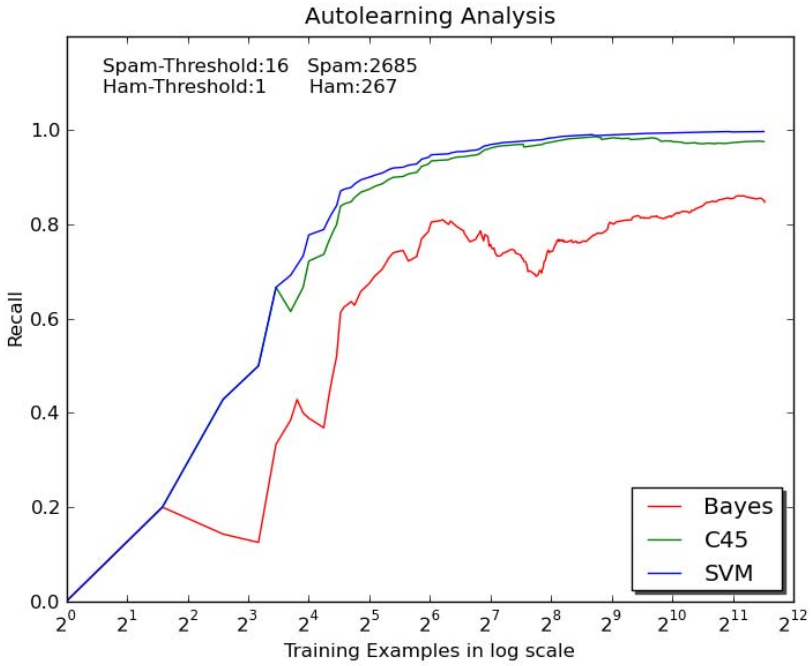

Figure 17. Auto-Learning (Threshold=16): Recall

We observe a gradual improvement in the performance in all metrics, with C4.5 and SVM achieving constant high rates above 95% in accuracy, precision, and recall with as few as 1,024 (2^{10}) training examples. Naïve Bayes shows high precision rates of 98% but low performance in accuracy and recall, probably due to the higher volume of spam-training examples. C4.5 has the second-best performance by achieving rates above 95% in accuracy and above 97% in precision and recall. Finally, SVM presents similarly high performance, with rates in accuracy and precision above 96% and above 99% in recall. Another noteworthy point is the behavior of all classifiers with respect to precision. With as few as 64 (2^6) training examples, all reached constant high precision rates above 95%. In the following experiments, we gradually increased the spam threshold, which resulted in fewer spam-training examples, and examples for which we have more confidence in their true disposition as spam.

Next, we increased the spam score threshold to 24, resulting in 960 spam-training examples, with ham examples being constant at 267, because we retained the same ham-score threshold.

	TP	FN	TN	FP
Bayes	2477	448	112	37
C45	2854	71	68	81
SVM	2917	8	36	113

Table 19. Auto-Learning (Threshold=16): Confusion Matrix

Naïve Bayes improved performance in all metrics, achieving rates above 92% in accuracy and recall compared to 84%, as shown in Figures 15, 16, and 17, and reached a 99% precision rate. C4.5 improved the performance in precision, with rates above 98%, but lowered slightly the rates in accuracy and recall, though still above 93% and 95%, respectively. Finally, SVM improved performance in all metrics, achieving rates above 97% in accuracy and precision and above 99% in recall. We show the results of our second auto-learning experiment in Figures 18, 19, and 20, and the respective confusion matrices in Table 20.

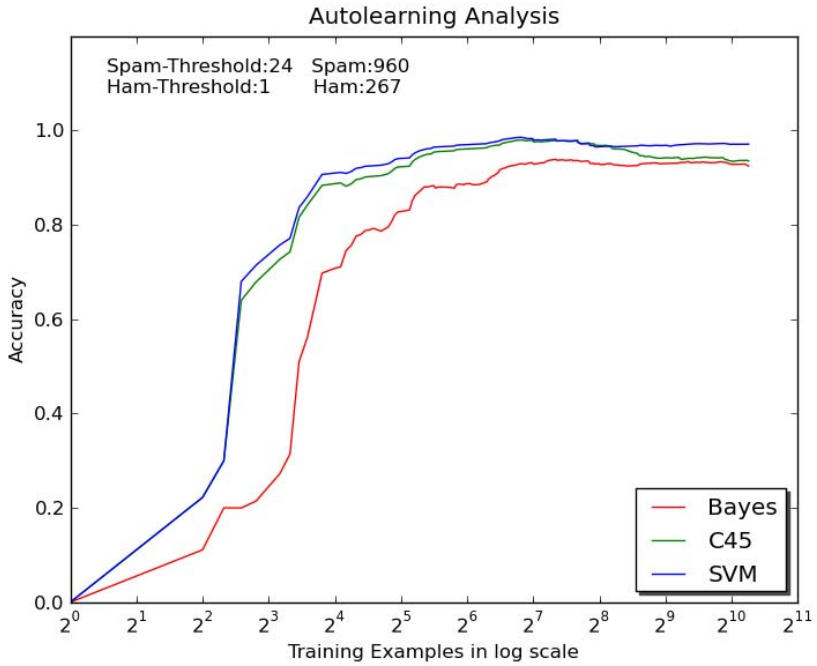

Figure 18. Auto-Learning (Threshold=24): Accuracy

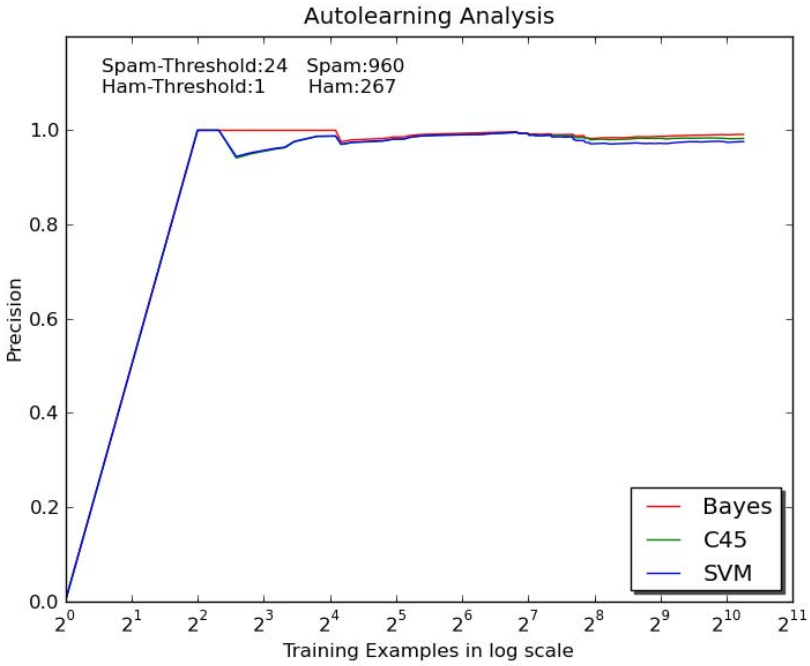

Figure 19. Auto-Learning (Threshold=24): Precision

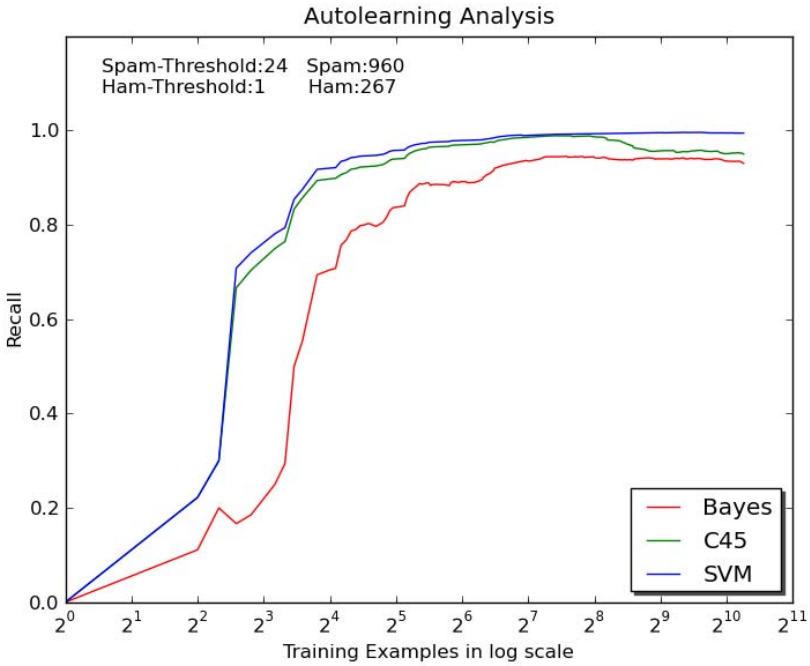

Figure 20. Auto-Learning (Threshold=24): Recall

	TP	FN	TN	FP
Bayes	4326	324	111	38
C45	4416	234	71	78
SVM	4625	25	36	113

Table 20. Auto-Learning (Threshold=24): Confusion Matrix

We increased the spam-score threshold in our third auto-learning experiment to 30 and the classifiers were trained with 229 spam flows and 267 ham flows. Figure 21 shows accuracy, and Figures 22, and 23, show precision and recall, respectively. Again, we observe a gradual improvement, but have slight differences in performance. Naïve Bayes shows the best behavior in all metrics, achieving high precision rates above 98% with as few as six training examples and improving gradually the performance in accuracy and recall, with constant rates above 95% after 165 training examples. C4.5 and SVM, on the other hand, show better performance in precision with respect to first and second experiments, achieving rates above 98% in accuracy and recall; however, their performance degrades, while still achieving rates above 90%. Table 21 shows the confusion matrix of this experiment.

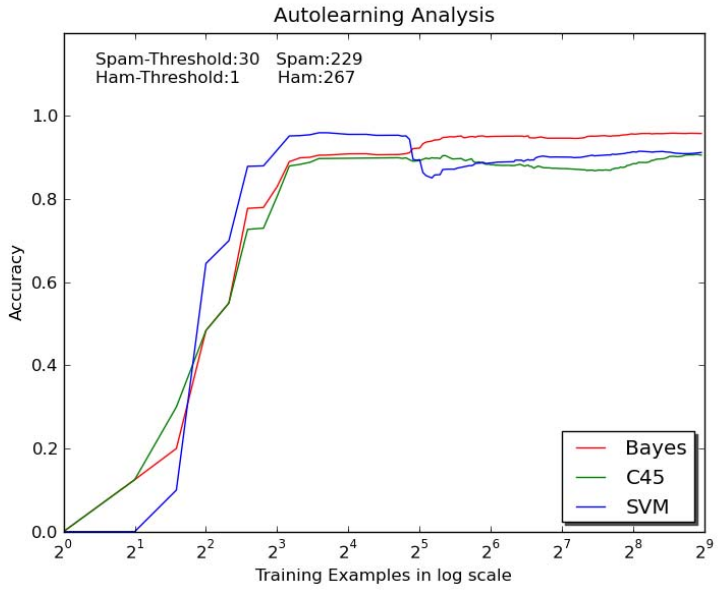

Figure 21. Auto-Learning (Threshold=30): Accuracy

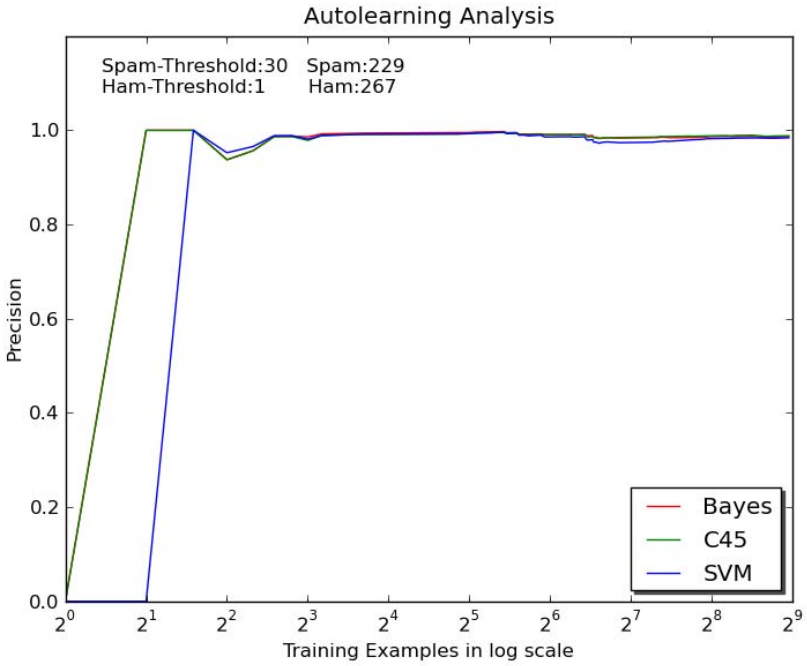

Figure 22. Auto-Learning (Threshold=30): Precision

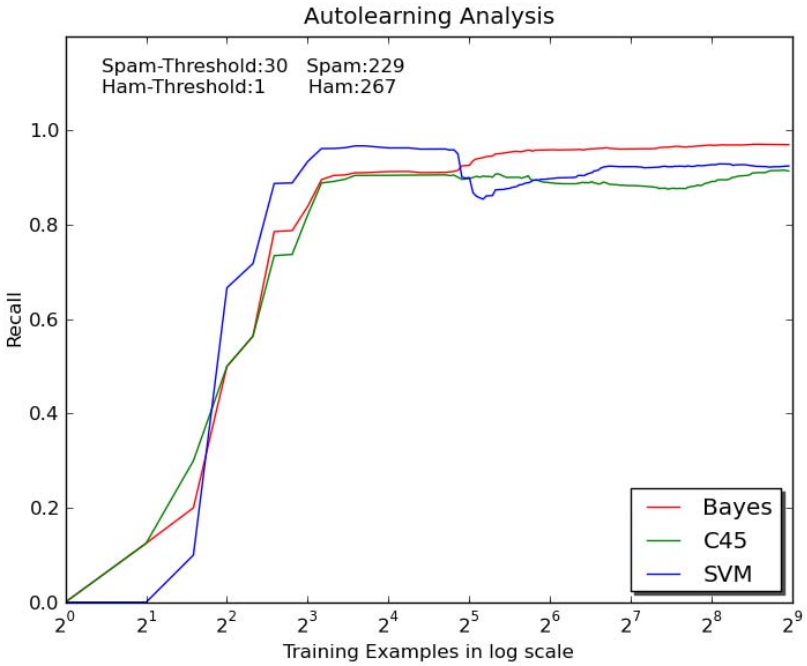

Figure 23. Auto-Learning (Threshold=30): Recall

	TP	FN	TN	FP
Bayes	5219	162	77	72
C45	4918	463	91	58
SVM	4975	406	69	80

Table 21. Auto-Learning (Threshold=30): Confusion Matrix

In the last experiment, we raised the spam score threshold to 40, and this ended up in training the classifiers with 30 spam training examples many fewer than the number of ham examples. The results are interesting as depicted in Figures 24, 25, and 26, and Table 22, with the precision rate remaining above 97% across all classifiers with as few as six training flows. C4.5 achieves the highest precision rate, with 99%, followed by SVM with 98% and naïve Bayes with 97%.

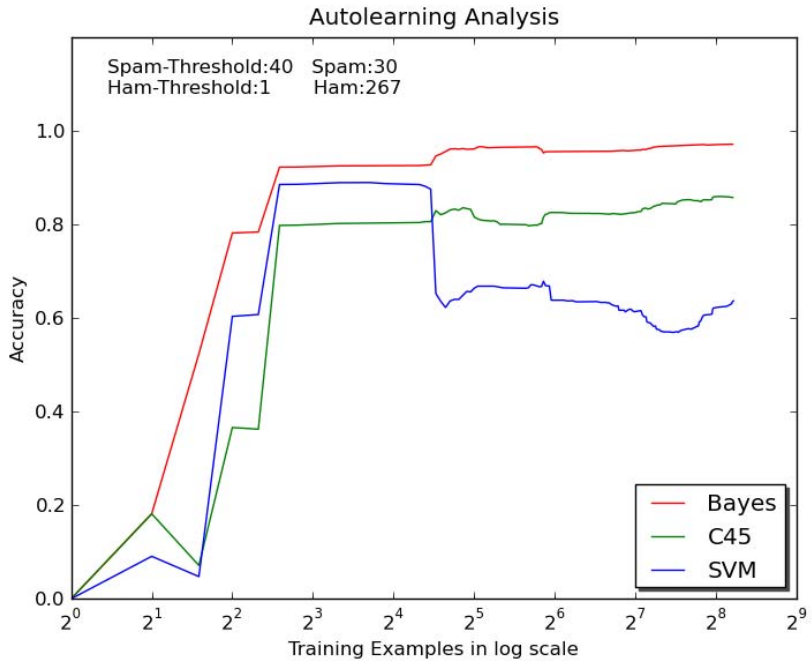

Figure 24. Auto-Learning (Threshold=40): Accuracy

The other interesting point is that naïve Bayes manages to retain high rates both in accuracy and recall, with rates of 97% and 99% respectively, even if we subtract spam-

training examples. Finally, the performance of both C4.5 and SVM degrades with SVM, falling to 60% after the point of 32 examples; but we have to keep in mind that we can tune SVM and find the kernel and parameters that best fit our features.

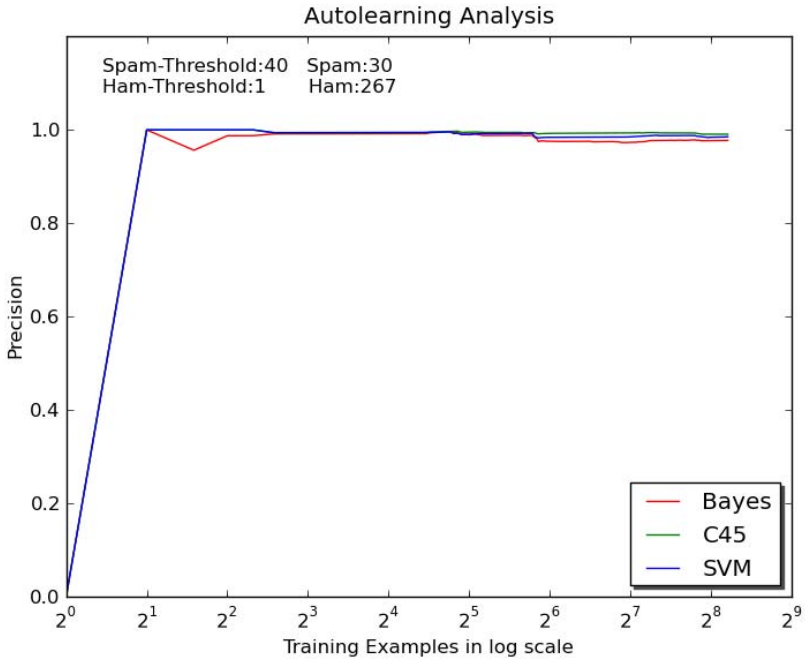

Figure 25. Auto-Learning (Threshold=40): Precision

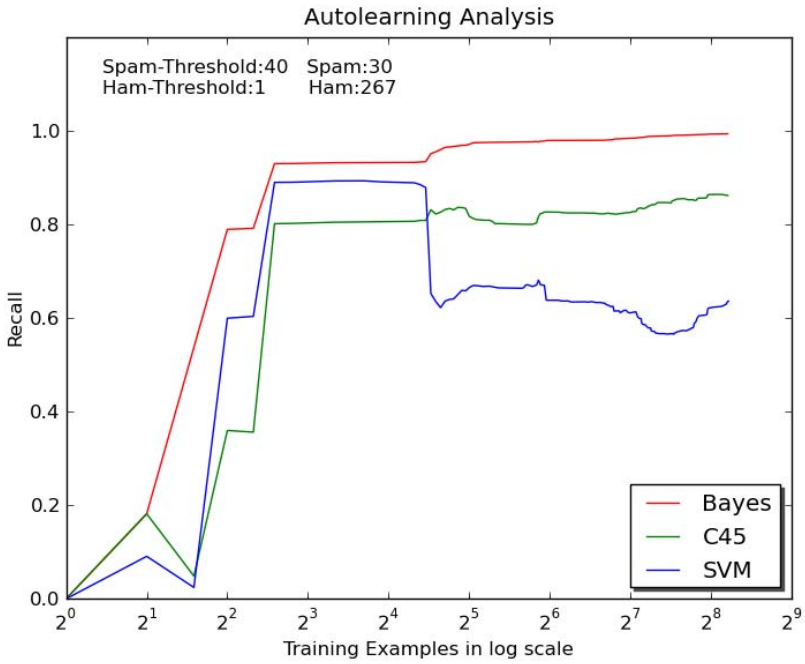

Figure 26. Auto-Learning (Threshold=40): Recall

	TP	FN	TN	FP
Bayes	5547	33	22	127
C45	4811	769	104	45
SVM	3553	2027	97	52

Table 22. Auto-Learning (Threshold=40): Confusion Matrix

V. CONCLUSIONS AND FUTURE WORK

The goal of this thesis was to develop and evaluate an online, real-time system for abusive-network-traffic detection, based on the previous work of Beverly and Sollins [3]. Our primary focus was to detect abusive traffic associated with unsolicited commercial e-mail. To accomplish our goal, we developed a test bed using virtual machines and a network emulator that induces generated SMTP traffic sufficient to match the characteristics that live spam traffic exhibits at the TCP layer. In Section A we discuss future work whereas in Section B our conclusions.

A. FUTURE WORK

Having gone through the process of building our test bed, deploying our system in live environment, and evaluating its performance in both environments, as well as in using the threshold-based, auto-learning mechanism, we discern a need for future work in system evaluation and application domains.

1. System Evaluation

Our test bed was not ideal for thorough system benchmarking. It served, however, as a step towards developing and integrating the different components and a first-phase evaluation. In future work, we would like to evaluate our system in a more realistic test bed like PlanetLab [62], where we could achieve higher data rates and observe the effects on throughput and system load. Furthermore, Dummynet uses an independent-loss model, which means that the decision to drop a packet is independent of whether a previous packet has been dropped. This model, however, does not represent the actual loss behavior that we experience in the Internet. Thus, we need to use a more representative model in the spirit of [63]. Since Dummynet is integrated into PlanetLab, PlanetLab should be an ideal testing platform. Moreover, we would like to extend our live testing by deploying our system in the network core, where we can experience high volumes of traffic and thus capture a more representative picture of spamming-network behavior at the TCP layer.

During our classification-engine evaluation process, we used a specific SVM kernel type; thus the results extracted may not be fully representative of SVM performance. SVM can be optimized by using different kernel types, and for each kernel type, different parameters. So, we would like to expand our system evaluation by using different kinds of kernel types and finding the best parameters to suit our features.

Furthermore, we would like to implement our auto-learning feature in *plugin*, since the experimental results were very promising, and then evaluate our system as a first line of defense using SpamAssassin for further analysis. We would like to expand on that by observing how *spamflow* can perform using unsupervised learning techniques, so that we do not have to select flows based on SpamAssassin scores. We would like to experiment on different techniques such as clustering, principal-component analysis, and independent-component analysis, and explore first whether we can have a strong separation between spam and ham features. Having established that, we would like to discover which spam flows exhibit strong entropy, in order to use them as training points for our classification engine.

We could further reinforce our simulated incremental-learning method to use actual incremental algorithms. These algorithms proceed in a sequence of trials and each trial is decomposed into three steps. First, the learning algorithm is presented with an example. The algorithm then predicts the label of the example; and finally, the algorithm is told the true label. The goal of these algorithms in the case of classification problems is to minimize the number of mistakes. This procedure is also called the mistake-bound learning model [64]. The main difficulty of online learning is the continuous requirement for label feedback, which in our case could be obtained using SpamAssassin.

Recently, several researchers have proposed incremental learning methodologies for spam detection [65, 66]. In addition, recent work on efficiently folding new positive samples into naïve Bayes [67, 68] and SVM [69, 70], is promising. We have shown that simulated incremental learning performs well using our SpamFlow techniques. Efficient, online, incremental learning will allow a fielded system to adjust to the dynamic threat environment.

2. Application Domains

Our goal was to develop a system for abusive network traffic detection, but we experimented specifically on abusive traffic originating from spammers. A botnet is a group of compromised hosts that are being controlled through a command-and-control mechanism. Botnet detection is an immediate application domain, since spammers, in order to obfuscate themselves, utilize botnets to send high volumes of traffic.

We would like to investigate, through statistical analysis, if received spam flows from botnets have a strong correlation and if spam traffic unrelated to botnets follows a more random distribution. If that is the case, we would be able to distinguish between originators of spam traffic. To accomplish this task, we could work in the same spirit as [71] and build a botnet infrastructure that would allow us not only to evaluate *spamflow* against spam-traffic detection, but against other types of abusive traffic that originate from botnets, such as denial-of-service attacks.

Second, botnets often host scam infrastructure. We wish to investigate whether we can detect the dual of the problem investigated in this thesis: if access to the bots, e.g., via a web-request, reveals similarly discriminating transport features.

A final possibility is to examine the effectiveness of *spamflow* in intrusion detection. To facilitate this task, we could deploy a honeypot or honeynet and integrate *spamflow* into Bro, an open-source network-intrusion and -detection system. Honeypots are decoy servers or systems that are being used in order to collect information about an attacker and honeynet is a collection of honeypots that form a network. Using statistical analysis such as clustering, correlation, or principal-component analysis, we would like to examine if we can discern different attacks based on TCP features and use Bro's estimations as our ground truth.

B. CONCLUSIONS

In developing the environment, we applied certain modifications to fit our needs. First, we modified the network emulator, dummynet, to produce a random delay with a mean μ and a standard deviation σ allowing us to simulate characteristics such as

congestion and variations in the roundtrip time (RTT). Secondly, we had to alter our MTA to include in the header of the message the IP and the port number of the sender. This was appropriate so that our flow-analysis engine could aggregate packets into flows, extract features, and match e-mails with flows. We also had to overcome a port-reusability problem, since the number of TCP connections that we had to establish demanded larger than the available range of ephemeral ports.

Further, we had to develop our classification engine and integrate our system into SpamAssassin so that it could operate in real-time. For the latter, we developed a plug-in that allowed us to obtain the information (IP:Port) we needed from SpamAssassin and use it as input into *spamflow*. The *plugin* was also responsible for the confusion matrices that we used during the evaluation process. We developed our classification engine using Orange, a statistical- and machine-learning software package. Finally, for communication between *plugin* and the *classifier* we used the standard and extensible XML-RPC protocol.

We evaluated our system in the test bed, as well as in a live, real-world environment. Our goal was to evaluate the performance in a high-rate environment and observe how it behaves in terms of throughput and system load. We achieved only moderate e-mail message-traffic rates, due to hardware restrictions, but the test bed was a first stage providing efficient high-rate testing. Next, we wanted to examine how strong TCP characteristics are in contrast with content features and we used SpamAssassin as a basis for comparison. Finally, we wanted to investigate how our system would perform in auto-learning mode. We defined auto-learning as the process of gradually training our system with flows that are associated with messages strongly indicative of being either spam or legitimate, as determined by SpamAssassin-assigned content and rule-based scores that match our threshold criteria.

Summarizing the results from our experiments, we present the following key points.

- All three classifiers achieved greater than 90% accuracy, precision, and recall in both the virtual test bed and in the live environment, which indicates that our system can adopt and capture any changes in the TCP characteristics.
- The high precision rate of 99% that all classifiers showed, along with the high accuracy, indicates the strong effectiveness of our system.
- The results showed that even with a small number of training examples, we can achieve great performance, which implies that the network behavior of spammers at the TCP level as captured by *spamflow* is quite discriminating and we need only a small number of training examples to initialize our system.
- Naïve Bayes performance, with respect not only to accuracy, precision, and recall, but also to throughput, makes it a viable candidate for our system.
- The performance of our system is relatively stable across all samples and independent of the classification method, which emphasizes the quality of *spamflow's* features and increases its reliability. Auto-learning showed that *spamflow* can achieve high rates in accuracy, recall, and precision. Especially in precision, all three classifiers achieved a 99% rate, suggesting that we could use *spamflow* as a first line of defense, and have suspicious messages further being filtered by SpamAssassin .

We hope that this thesis serves to sufficiently motivate our approach to abusive-traffic detection and mitigation and to warrant further research in the area.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Baracuda Networks. Spam central: Spam emails detected. Available: http://www.barracudanetworks.com/ns/resources/spam_central.php.

[2] M. Carbone and L. Rizzo, "Dummynet Revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 12–20, April 2010.

[3] R. Beverly and K. Sollins, "Exploiting transport-level characteristics of spam," in *CEAS 2008 - Fifth Conference on Email and Anti-Spam*.

[4] J. Klensin. (2001, April). Simple mail transfer protocol. *Internet RFC 2821 (Standards Track)* Available: http://www.ietf.org/rfc/rfc2821.txt.

[5] D. Crocker, "Mail transfer agent," in *Internet RFC 5598-Internet Email Architectur*e, p. 31.

[6] Postel. (1981, September). Transmission control protocol. *Internet RFC 793* Available: http://www.ietf.org/rfc/rfc793.txt.

[7] J. F. Kurose and K. W. Ross, "Electronic mail in the internet." in *Computer Networking:A Top Down Approach*., 5th ed. Anonymous Addison Wesley, 2010, p. 120.

[8] Messaging Anti-Abuse working Group (MAAWG), "Email metrics program: The network's operator perspective," Tech. Rep. 13, November. 2010. Available: http://www.maawg.org/sites/maawg/files/news/MAAWG_2010-Q1Q2_Metrics_Report_13.pdf.

[9] C. Drake, J. Oliver and E. Koontz, "Anatomy of a phishing email," in *Proceedings of the First Conference on Email and Anti-Spam.* 2004.

[10] M. Siponen and C. Stucke, "Effective anti-spam strategies in companies: An international study," in *Proceedings of the 39th Hawaii International Conference on System Sciences.* 2006.

[11] The Spamhaus Project. Spamhaus. Available: http://www.spamhaus.org/.

[12] MAPS. Introduction to the realtime blackhole list (RBL). Available: http://www.mail-abuse.com/pdf/WP_MAPS_RBL_060104.pdf.

[13] J. Postel. (1981, September). Internet protocol. Internet RFC 791. Available: http://www.faqs.org/rfcs/rfc791.html.

[14] R. Thomas and D. Samoseiko, "The game goes on: An analysis of modern spam techniques," in *Virus Bulletin Conference,* Montreal, Canada, 2006.

[15] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers." in *Proceedings of ACM SIGCOMM.*

[16] S. Hao, N. A. Syed, N. Feamster, A. G. Gray and S. Krasser, "Detecting spammers with SNARE: Spatio-temporal network-level automatic reputation engine." *In Proceedings of the 18th Conference on USENIX Security Symposium.*

[17] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artif. Intell. Rev.,* vol. 29, pp. 63–92, March 2008.

[18] M. Sahami, S. Dumais, D. Heckerman and E. Horvitz, "A Bayesian Approach to Filtering Junk E-Mail," *AAAI Technical Report WS-98-05,* 1998.

[19] H. Drucker, Donghui Wu and V. N. Vapnik, "Support vector machines for spam categorization," *Neural Networks, IEEE Transactions,* vol. 10, pp. 1048–1054, 1999.

[20] R. O. Duda and P. E. Hart, "Bayes decision theory," in *Pattern Classification and Scene Analysis*Anonymous, John Wiley & Sons, 1973, p. 10.

[21] T. M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[22] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras and C. D. Spyropoulos, "An evaluation of naive bayesian anti-spam filtering," in *Proceedings of the Workshop on Machine Learning in the New Information Age.* Barcelona, Spain, 2000, pp. 9–17.

[23] V. N. Vapnik, "Estimation of Dependencies Based on Empirical Data," 1992.

[24] J. R. Quinlan, "C4.5: Programs for Machine Learning," 1993.

[25] S. Russell and P. Norvig, *Artificial Intelligence. A Modern Approach.* New Jersey: Pearson Education Inc., 2003.

[26] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning,* 1986.

[27] E. Blanzieri and A. Bryl, "Evaluation of the highest probability SVM nearest neighbor classifier with variable relative error cost," in *Proceedings of the Fourth Conference on Email and Anti-Spam.*

[28] F. Roli, B. Biggio, G. Fumera, I. Pillai and R. Satta, "Image spam filtering by detection of adversarial obfuscated text."

[29] A. Gray and M. Haahr, "Personalized, collaborative spam filtering," in *Proceedings of First Conference on Email and Anti-Spam*.

[30] V. V. Prakash. Vipul's razor. Available: http://razor.sourgeforge.net.

[31] D. Alperovitch, P. Judge and S. Krasser, "Taxonomy of email reputation systems," in *27th International Conference on Distributed Computing Systems Workshop (ICDCSW'07).*

[32] V. V. Prakash and A. O'Donnel, "Fighting Spam with Reputation Systems," *ACM QUEUE,* November 2009.

[33] Rhyolite. (2011, February 2). Distributed checksum clearinghouses (DCC). Available: http://www.rhyolite.com/dcc/.

[34] J. Lyon and M. Wong. (2006, Sender ID: Authenticating E-mail. *Internet RFC 4406 (Experimental).* Available: http://www.ietf.org/rfc/rfc4406.txt.

[35] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton and M. Thomas. (2007, May). DomainKeys identified mail (DKIM) signatures. *Internet RFC 4871 (Standards Track).* Available: http://www.ietf.org/rfc/rfc4871.txt.

[36] B. Leiba, J. Ossher, V. T. Rajan, R. Segal and M. N. Wegman, "SMTP path analysis," in *Proceedings of Second Conference on Email and Anti-Spam.* 2005.

[37] J. Goldbeck and J. Handler, "Reputation network analysis for email filtering," in *Proceedings of First Conference on Email and Anti-Spam.* 2004.

[38] R. Haskins, "The Rise of Reputations in the Fight Against Spam." Available: http://linux.sys-con.com/node/48128.

[39] J. Karlin, S. Forest and J. Rexford, "Autonomous security for autonomous systems." *Computer Networks,* vol. 52, pp. 2908–2923, 2008.

[40] X. Zhao, D. Pei, L. Wang, D. Massey and A. Mankin, "An analysis of BGP multiple origin AS (MOAS) conflicts," in *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement (IMW),* 2001.

[41] T. Ouyang, S. Ray, M. Allman and M. Rabinovich, "A Large-Scale Empirical Analysis of Email Spam Detection through Transport-level Characteristics," *Technical Report 10-001, International Computer Science Institute.* January 2010.

[42] H. Esquivel, T. Mori and A. Akella, "Router-level spam filtering using TCP fingerprints: Architecture and measurement-based evaluation," in *Sixth Conference on Email and Anti-Spam*, 2009.

[43] M. Zalewski. (2006, The new P0f. Available: http://lcamtuf.coredump.cx/p0f.shtml.

[44] D. Schatzmann, M. Burkhart and T. Spyropoulos, "Inferring spammers in the network core," in *Passive and Active Conference*, Seoul,South Corea, 2009, pp. 229–238.

[45] Internet Systems Consortium. BIND: Berkeley internet name domain. Available: http://www.isc.org/software/bind.

[46] W. Venema. Postfix. Available: http://www.postfix.org/documentation.html.

[47] J. Mason. Filtering spam with SpamAssassin. Presented at HEANet Annual Conference. Available: http://wiki.apache.org/spamassassin/PresentationsAndPapers.

[48] V. Jacobson, C. Leres and S. McCanne. (2008, October 27, 2010). Packet capture library (pcap). Available: http://www.tcpdump.org/pcap3_man.html.

[49] Laboratory of Artificial Intelligence, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, "Orange: A Component Based Machine Learning Library for Python," vol. 2.0, 2010.

[50] G. Cormack and T. Lynam. 2005 TREC public spam Corpus.

[51] G. Van Rossum. Python. Available: http://www.python.org/.

[52] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *ACM Computer Communication Review*, vol. 27, pp. 31–41, 1997.

[53] F. Tobin, "Pyzor: Spam-Blocking Networked System," Available: http://sourceforge.net/projects/pyzor/.

[54] V. Jacobson, C. Leres and S. McCanne, "Packet Capture Library (pcap)," vol. 1.0.0, October 27, 2010.

[55] Baracuda Networks. (2003, October). Baracuda spam and virus firewall. Available: http://www.barracudanetworks.com/ns/products/spam_overview.php.

[56] W. Larry. Perl. Available: http://perldoc.perl.org/.

[57] D. Winer. (1998, April). XML-RPC specification. Available: http://www.xmlrpc.com/spec.

[58] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau. (2008, November 26). Extensive markup language (XML) 1.0. *W3C Recommendation* Available: http://www.w3.org/TR/2008/REC-xml-20081126/.

[59] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. (1999, June). Hypertext transfer protocol--HTTP/1.1. *Internet RFC 2616 (Standards Track)* Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[60] S. A. Alvarez, "An exact analytical relation among recall, precision and classification accuracy in information retrieval," Computer Science Department, Boston College, Tech. Rep. BC-CS-02-01, July 2002.

[61] C. Chang, C. Chang and C. Lin, "A Practical Guide to Support Vector Classification," April 15, 2010.

[62] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniac and M. Bowman, " PlanetLab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 3–4, 12, 2003.

[63] E. M. Nahum, M. Rosu, S. Seshan and J. Almeida, "The effects of wide-area conditions on WWW server performance," *SIGMETRICS Perform.Eval.Rev.*, vol. 29, pp. 257–267, June 2001.

[64] A. Blum, "On-Line Algorithms in Machine Learning (a survey)." *This is a Survey Paper for a Talk Given at the Dagstuhl Workshop on on-Line Algorithms*, June 1996.

[65] C. Siefkes, F. Assis, S. Chhabra and W. S. Yerazunis, "Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering," vol. 3202, pp. 410–421, 2004.

[66] V. Keiser and T. G. Dietterich, "Evaluating online text classifications algorithms for email prediction in TaskTracer," in *Sixth Conference on Email and Anti-Spam CEAS 2009*, Mountain View, California, 2009.

[67] L. Fei-Fei, R. Fergus and P. Perona, "Learning generative visual models from few training examples: An Incremental Bayesian approach tested on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, pp. 59–70, April 2007.

[68] M. Godec, C. Leistner, A. Saffari and H. Bischof, "On-line random naive bayes for tracking," in *Proceedings of the 20th International Conference on Pattern Recognition (ICPR).* Instanbul, 2010, pp. 3545–3548.

[69] G. Cauwenberghs and T. Poggio, "Incremental and Decremental Support Vector Machine Learning," *Advances in Neural Information Processing Systems,* vol. 13, 2001.

[70] C. Tseng and M. Chen, "Incremental SVM model for spam detection on dynamic email social networks," in *International Conference on Computational Science and Engineering,* Vancouver,Canada, 2009, pp. 128–135.

[71] W. T. Strayer, D. Lapsley, R. Walsh and C. Livadas, "Botnet Detection Based on Network Behavior." *In Botnet Detection: Countering the Largest Security Threat,* 2007.

INITIAL DISTRIBUTION LIST

3. Defense Technical Information Center
   Ft. Belvoir, Virginia

4. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

5. Hellenic Navy General Staff
   Athens, Greece

6. Hellenic Fleet Command
   Salamis, Greece